Towards Creating Context-Aware Dynamically-Adaptable Business Processes Using Complex Event Processing

Gabriel Hermosillo

Département de formation doctorale en informatique **École doctorale SPI Lille**
UFR IEEA

Towards Creating Context-Aware Dynamically-Adaptable Business Processes Using Complex Event Processing

THÈSE

présentée et soutenue publiquement le 5 Juin 2012

pour l'obtention du

Doctorat de l'université Lille 1 Sciences et Technologies

(spécialité informatique)

par

Gabriel Hermosillo

Composition du jury

Rapporteurs :	Françoise Baude – *Professeur, Université de Nice-Sophia Antipolis – France*
	Didier Donsez – *Professeur, Université Grenoble 1 – France*
Examinateurs :	Franck Barbier – *Professeur, Université de Pau et des Pays de l'Adour – France*
	Luigi Lancieri – *Professeur, Université Lille 1 – France*
Directeurs :	Laurence Duchien – *Professeur, Université Lille 1 – France*
	Lionel Seinturier – *Professeur, Université Lille 1 et IUF – France*

Laboratoire d'Informatique Fondamentale de Lille – UMR USTL/CNRS 8022 – INRIA Lille Nord Europe

Abstract

As the use of ubiquitous devices continues to grow, we have more and more access to pervasive information around us. This information allows us to know the state of our surroundings, and we make decisions of our everyday life based on that context information. Computer-based business processes are expanding more and more, as the activities they deal with are being automatized. However, when dealing with these processes, there is a lack of integration between the context information and the processes.

We can consider a condition in a specific part of the process and make a decision based on the information, but we cannot monitor the information in real-time and adapt the process accordingly, just as we do in normal life. Moreover, the static nature of business processes does not allow them to be dynamically modified, thus leaving them less useful in the new context. If we want to change the behavior of a business process, we need to stop it, modify it and redeploy it entirely, which causes to lose all the current executions and information.

To address these issues, in this thesis we present the CEVICHE Framework. We bring forward an approach which allows to represent context-aware business processes where context information is considered as events which are monitored in real-time. For this, we base our work on a technique called Complex Event Processing (CEP). By using an external tool to monitor the context in real-time, we are able to surpass the limit of only accessing the information on specific places of the process. However, knowing this information is not enough, as what we need is to be able to adapt our processes accordingly.

With CEVICHE we integrate the information obtained from the context with the capability of adapting business process at run-time. Also, one of the original contributions of the CEVICHE Framework is the definition of a correct adaptation undoing mechanism and its implementation. Undoing an adaptation can easily go wrong and lead to undesired states and unstable processes. Naively considered as a trivial task, this issue has been barely considered when looking at the current dynamic approaches, and in the business process domain, none of the approaches

integrates this. So, in CEVICHE we propose a formal model of this mechanism for undoing adaptations.

The implementation of the CEVICHE Framework offers flexibility and dynamicity properties to the business processes, using a component-based approach, allowing the modification of their bindings at run-time. Moreover, with CEVICHE we also provide a stability property in terms of CEP. As any new technology, CEP is still evolving and since there is still no standard in the way in which the events are defined, each implementation uses its own language to express them. By defining our own simple language, the *Adaptive Business Process Language* (ABPL), as a pivot language, CEVICHE facilitates the use of CEP without the drawbacks of early adoption. We use a plug-in approach that allows the events defined in ABPL to be used in virtually any CEP engine. This approach also makes it easier to maintain, as we just need to update the plug-in in case the CEP language evolves, or we decide to use another implementation, instead of updating all the event definitions.

Finally, we validated our approach by implementing a nuclear crisis scenario, with a use case which involves many actors, context-information and adaptation conditions.

Résumé

En plus de l'utilisation des appareils ubiquitaires qui continue à croître, nous avons accès de plus en plus à d'informations dites contextuelles. Ces informations permettent de connaître l'état de notre environnement et nous aident à prendre les décisions de notre vie quotidienne en fonction du contexte dans lequel nous nous positionnons. Les processus métiers informatiques sont de plus en plus en expansion, alors que les activiés qu'ils traitent deviennent automatisées. Toutefois, lorsqu'il s'agit de processus métiers dans un domaine particulier, il y a un manque d'intégration avec ces informations contextuelles.

Nous pouvons envisager actuellement une situation donnée dans une partie bien définie du processus à un moment donné et prendre une décision basée sur cette information, mais nous ne pouvons pas contrôler ces informations contextuelles en temps réel et adapter le processus en conséquence, comme nous le faisons dans la vie normale. De plus, la nature statique des processus métiers ne leur permet pas d'être modifiés dynamiquement, les rendant ainsi moins utiles dans un nouveau contexte. Si nous voulons changer le comportement d'un processus métier, nous devons le stopper, le modifier et le redéployer entièrement, ce qui fait perdre toutes les exécutions en cours et l'information associée.

Pour répondre à ces problèmes, dans cette thèse, nous présentons le cadre logiciel CEVICHE. Nous proposons une approche qui permet de représenter des processus métiers sensibles au contexte où les informations de contexte sont considérées comme des événements contrôlés en temps réel. Pour cela, nous nous basons sur une nouvelle approche appelée *Complex Event Processing* (CEP). En utilisant un outil externe pour contrôler le contexte en temps réel, nous sommes alors en mesure de dépasser les limites d'accés à l'information uniquement à des endroits bien précis du processus. Cependant, la connaissance de ces événements ne suffit pas. Nous avons, de plus, besoin d'être capable d'adapter nos processus en conséquence à l'exécution.

Avec CEVICHE, nous intégrons les informations obtenues à partir du contexte avec la capacité d'adaptation des processus métiers en cours d'exécution. De plus, l'une des originalités du cadre logiciel CEVICHE vient de la définition d'une opération de désadaptation et de sa mise en œuvre. Défaire l'adaptation peut facilement

se passer mal et conduire à des états non désirés et rendre les processus instables. Naïvement considérée comme une tâche triviale, cette question a été peu considérée quand on regarde les approches dynamiques actuelles. Nous proposons donc un modèle formel de ce mécanisme dans CEVICHE.

La réalisation du cadre logiciel CEVICHE offre des propriétés de flexibilité et de dynamicité aux processus métiers en se basant sur une approche à composants, permettant ainsi la modification des liaisons en cours d'exécution. En outre, avec CEVICHE, nous apportons une propriété de stabilité au niveau du traitement des événements complexes. Comme toute nouvelle approche, le traitement des événements complexes n'est pas normalisé et est en cours d'évolution, chaque outil utilisant son propre langage pour les exprimer.

En définissant notre propre langage, *Adaptive Business Process Language* (ABPL), comme un langage pivot, CEVICHE facilite l'utilisation de CEP sans les inconvénients de l'adoption anticipée de l'approche. Nous utilisons une technique de type plug-in qui permet aux événements définis en ABPL d'être utilisés dans pratiquement n'importe quel moteur CEP. Cette approche rend les règles de traitement des événements plus faciles à maintenir, car nous centralisons la mise à jour au niveau du plug-in lorsque le langage CEP évolue, ou si nous décidons l'utilisation d'un autre outil, au lieu de mettre à jour toutes les définitions d'événements.

Finalement, nous avons validé notre approche en mettant en œuvre un scénario de crise nucléaire, avec des cas d'utilisation qui impliquent de nombreux acteurs, des informations de contexte et des conditions d'adaptation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"If Tetris has taught me anything, it is that errors pile up and accomplishments disappear."
- Anonymous

Contents

1.1 Introduction

In order to maintain a competitive level, organizations are increasingly using service-oriented solutions to automate and facilitate the integration of their business processes. These solutions rely mainly on well known standards, such as BPEL[1] (*Business Process Execution Language*), to orchestrate the services during the execution of the process. As the business processes evolve and become more complex, the data around them increases exponentially, and there are more and more

[1] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

factors that can affect the correct performance or the even successful execution of the process.

The information of these factors around the business process execution is known as context information [Dey et al., 2001]. Given that this information is constantly changing, it is important to be able to monitor it, in order to identify when special situations are developing, which could affect the execution of the process. Unfortunately, the BPEL specification does not provide any standard mechanisms to monitor context information. Although this could be done by adding monitoring activities throughout the process definition, it would only result in a narrow solution which would give specific context information at specific parts of the process, besides creating more problems of its own (*e.g.*, maintainability, cross-cutting concerns, etc.).

Nevertheless, there are some techniques that would allow the monitoring of this information without interfering with the execution of the process. One of these techniques is Complex Event Processing (CEP). CEP is an emerging technology that can help the organizations to benefit from context information, since it allows them to find real-time relationships between different events, using elements such as timing, causality, and membership in a stream of data to extract relevant information [Luckham, 2002]. CEP is used in a wide variety of applications, like preventing theft of merchandise [Huber and Michael, 2007], monitoring the stock market [Mangkorntong, 2009], and interacting with RFID systems [Zang et al., 2008].

However, monitoring context information to identify special situations is not enough, since we still need to be able to respond to those situations. Business process definitions are static by nature, which means that they cannot be modified at run-time. At the same time, there is a strong need to be able to adapt those processes dynamically, in order to respond to the changing conditions.

Changing the processes manually takes considerably more time, and is more error-prone. Moreover, redeploying a modified business process would lead to downtime of the service and loss of information of all the currently executed instances of the process. In this dissertation we explore the reconfiguration capabilities provided by the Service Component Architecture (SCA) as a solution to this problem [Beisiegel et al., 2007]. Our proposal intends to integrate the benefits from CEP to monitor context information, as well as the reconfiguration capabilities of SCA into existing business processes.

Structure of the Chapter

The remainder of this introductory chapter is organized as follows: In Section 1.2 we identify the problems that motivate this research. Next, in Section 1.3 we present our research goals. Section 1.4 explains the contributions made by this dissertation. In Section 1.5 we give a brief introduction to each of the chapters of the document. Finally, in Section 1.6 we list the publications made during the development of our work.

1.2 Problem Statement

As we explained before, there are currently many problems that hinder the successful execution of business process, specially when considering the high dynamicity of the environments in which they have to be executed. During this dissertation we have identified the following issues:

Difficulty to integrate context information

Business processes lack the capabilities to constantly monitor the information around them. They have a limited scope on the information they can get, and the moment when they have access to it. Nowadays, thanks to the growth of pervasive and ubiquitous environments, we have access to more and more information that could be useful to the execution of the processes, but from which they cannot benefit because of the restraints of the specification.

Static nature of business processes

Besides their lack of monitoring capabilities, business processes are static by nature, which means that they cannot be modified at run-time. They were not meant to be dynamically changed, as they were thought simply as a predefined sequence of activities to achieve a goal. There is actually certain flexibility in terms of changing the service providers, if they are specified beforehand in the business process definition, but we cannot change the behavior of the process as such (add new activities or remove existing ones).

Adopting an early technology

We previously mentioned using CEP as a solution to monitoring context information. However, this is an early technology, and as such it is still prone to constant evolution. CEP has attracted the attention of several developers on different domains, in both research and industry, which has lead to the creation of many solutions implementing it. But there is no standard as of how to define the rules that the user feeds to the CEP engine, and each implementation uses its own language. So, how can we take early advantage of the benefits of CEP without having to deal with the drawbacks of a still immature and evolving technology?

Unadapting could be dangerous

The final problem is not specific to business process, but actually a common problem to all dynamic adaptation solutions. Once we are able to handle dynamic adaptation, we may also need to later undo such adaptation because the adapting conditions are no longer valid. This is usually seen as an easy task, however, if it is not done in a proper manner, it can lead to inconsistent states of the application, or in this case, business process.

1.3 Goals of this dissertation

Given the problems presented in the previous section, the goals of this dissertation are focused on bringing a solution to them. With our work we plan to improve the execution of business processes by providing them with **context-awareness**, **dynamic adaptation**, and **correct and automatic undoing of adaptations**. At the same time, we want to provide the user with a **platform independent** solution that allows the user the flexibility to choose the best CEP engine she considers optimal, without having to worry about the different languages or implementations in the market.

Context-awareness

In order to ameliorate the deafness relating to context information that currently exists in business processes, we will integrate CEP as an external source of information, that will constantly monitor the changes in the environment and detect

when an adaptation is needed in the process to continue an optimal execution. As an external source, it will not affect the performance nor the maintainability of the original process, and will provide an easier way to update the situations that we want to monitor.

Dynamic adaptation

A fundamental part of our proposal is dynamicity, and by it we mean to be able to automatically adapt the business processes at run-time, without any downtime nor loss of information. We need to be able to rapidly respond to the changes in the context in order to guarantee that our process is executed under the best possible conditions to achieve its goal.

Platform independence

Choice is a very valuable asset. The possibility to be able to change from one option to another, specially when the marketplace of those options is in continuous change while achieving maturity, is extremely important. In the case of CEP, the change from one solution to another would imply to actually redo all the defined rules for monitoring in a new language, which sometimes has a completely different syntax.

Correct and automatic undoing of adaptations

When dealing with context-awareness and dynamic adaptation, there is something that we must also consider, and that is to be able to undo the changes done to the business process when the adaptation conditions are no longer valid. This is not a simple nor trivial task, as all subsequent and dependent adaptations must also be considered during the process of undoing the adaptation.

1.4 Contribution

In order to enhance the understanding of our work, in this section we briefly describe the main contributions of this dissertation. As stated before, the goal of our

work is to provide the user with context-aware dynamically adaptable business processes. To achieve that we created the CEVICHE framework.

Our first contribution is an adaptive language, called the *Adaptive Business Process Language* (**ABPL**), that allows the user to define the adaptation needs, by responding to the four adaptation questions: *What* to adapt?, *When* to adapt it?, *Where* to adapt?, and *How* to adapt it? The ABPL merges adaptation and event definitions to simplify its use for a business process.

The second contribution of this dissertation is to provide a **Plug-in approach** that will allow the CEVICHE framework to interact with any existing and future CEP engine for context integration. A plug-in uses the information provided by the user in the ABPL and translates it to an engine-specific language, which prevents the user from having to rewrite all the rules related to finding an adaptation situation.

The third, and probably the most important contribution of our work is an **Adaptation Manager** built in the CEVICHE framework. It handles the actual manipulation of the business process and considers the information coming from the context monitors to trigger the adaptations. It stores the adaptation information in order to be able to identify when an adaptation condition is no longer valid and trigger its undoing. Finally, it manages the undoing of adaptations considering also the other adaptations that were done afterwards to look for related adaptations that could have happened.

1.5 Organization of the document

This dissertation is divided in four parts. The first part, State of the Art, gives the bases of the domain in which our work takes place, and analyzes some related work. The second part, Contribution, presents our work in more detail. In the third part, Validation, we present some measures and the implementation of a use-case scenario. In the final part, Conclusion, we summarize our work and discuss some perspectives.

Part I: State of the Art

Chapter 2: Background and concepts. In this chapter we give a brief introduction to some of the domains used throughout the dissertation, to allow a better under-

standing of the background and context in which our work takes place, as well as the terminology and concepts presented in the later chapters.

Chapter 3: Business Processes and Adaptation. In this chapter we present several works that have been trying to solve the lack of flexibility on business processes with different approaches. We compare these approaches using different criteria related to the type of adaptation that they use and the mechanisms to achieve it.

Part II: Contribution

Chapter 4: Event-based Dynamically-adaptable Business Processes. In this chapter we present our solution for dynamically adapting business processes using an event-driven approach to provide context information. We also show how undoing these adaptations is not a trivial task and present our proposal for correctly undoing an event-based adaptation, in a clean and automatic way.

Chapter 5: The CEVICHE Framework. In this chapter we present our implementation for doing and undoing dynamic adaptations, called the CEVICHE Framework. We use a component-based approach to provide dynamicity to business processes and Complex Event Processing as a way to deal with context information. Finally, we also present our adaptive language, the ABPL, and show an example of the use of the plug-in approach.

Part III: Validation

Chapter 6: Validation. In this chapter we validate our work by using a nuclear crisis management scenario, and we present the results of our tests to demonstrate why dynamic adaptations is the best solution and how its overhead can be negligible.

Part IV: Conclusion

Chapter 7: Conclusions and Perspectives. In this chapter we present the conclusions of our work and present some short-term and long-term perspectives.

1.6 Publications

Below we present a list of the research publications that were created during the development of our work around this dissertation.

International Journals

- Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy and Frank Eliassen. *The DigiHome Service-Oriented Platform*. Software: Practice and Experience (SP&E). 2012. Pages 17. *To appear*.
  Rank (CORE): A

- Gabriel Hermosillo, Sébastien Mosser, Lionel Seinturier, Laurence Duchien. *CEVICHE: A Framework for Dynamically Doing and Undoing Adaptations in Context-Aware Business Process with an Event-Driven Approach*. Journal of Systems and Software (JSS). 2012. Pages 25. *Submitted*.
  Rank (CORE): A

International Conferences

- Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle, Lionel Seinturier. *A Middleware Platform to Federate Complex Event Processing*. The Sixteenth IEEE International EDOC Conference (EDOC'12). Pages 10. Beijing, China. September 2012. *To appear*.
  Rank (CORE): B

- Sébastien Mosser, Gabriel Hermosillo, Anne-Françoise Le Meur, Lionel Seinturier, Laurence Duchien. *Undoing Event-Driven Adaptation of Business Processes*. The 8th IEEE 2010 International Conference on Services Computing (SCC'11). Pages 234–241. Washington D.C., USA. July 2011.
  Acceptance rate: 17%, Rank (CORE): A

- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Creating Context-Adaptive Business Processes*. The 8th International Conference on Service Oriented Computing (ICSOC'10). Pages 228–242. San Francisco, Calif., USA. December 2010.
  Acceptance rate: 15%, Rank (CORE): A

- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Using Complex Event Processing for Dynamic Business Process Adaptation*. The 7th IEEE International Conference on Services Computing (SCC'10). Pages 466–473. Miami, Florida, USA. July 2010.
  Acceptance rate: 18%, Rank (CORE): A

- Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy and Frank Eliassen. *RESTful Integration of Heterogeneous Devices in Pervasive Environments*. 10th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'10). Pages 1–14. Amsterdam, Netherlands. June 2010.
  Acceptance rate: 32%, Rank (CORE): B

- Patricia Jaimes, Gabriel Hermosillo, Roberto Gómez. *Una marca de agua inteligente aplicada al dinero electrónico*. The Fifth Ibero-American Congress on Information Security (CIBSI'09). Pages 225–239. Montevideo, Uruguay. November 2009.

International Workshops

- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Complex Event Processing for Context-Adaptive Business Processes*. The 8th BElgian-NEtherlands software eVOLution seminar (BENEVOL'09). Louvain-la-neuve, Belgium. December 2009.

- Gabriel Hermosillo, Julien Ellart, Lionel Seinturier, Laurence Duchien. *A Traceability Service to Facilitate RFID Adoption in the Retail Supply Chain*. The 3rd International Workshop on RFID Technology - Concepts, Applications, Challenges (IWRT'09). Pages 49–58. Milan, Italy. May 2009.

Posters

- Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle, Lionel Seinturier. *Distributed Complex Event Processing Engine*. Génie de la Programmation et du Logiciel (GPL'12). Rennes, France. June 2012.

- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Using CEP to create context-adaptive processes in pervasive environments*. CANOE and EuroSys Summer School. Oslo, Norway. August 2009.

Part I

State of the Art

Chapter 2

Background and concepts

"Don't ask what it means, but rather how it is used."
- Ludwig Wittgenstein

Contents

2.1 Introduction

This dissertation aims at adding dynamicity and flexibility to business processes. Flexibility in the sense of allowing the business processes to be adapted to the context around them, and dynamicity in the sense of the capability to make that adaptation automatically at run-time. To achieve such a goal, we make use of several technologies and approaches that facilitate this task. The following chapters will serve to set a base for the reader in terms of such technologies, presented in this Chapter, as well as to compare some of the related work around business process adaptation, which we present in Chapter 3.

The objective of this first chapter of the *State of the Art* is not to present an in-depth description of all the existing technologies surrounding business process adaptation, but to give a brief introduction to some of the domains used throughout the dissertation, to allow a better understanding of the background and context in which our work takes place, as well as the terminology and concepts presented in the later chapters.

Structure of the Chapter

The remainder of this chapter is divided in two main parts: i) In the first part, Section 2.2, we decided to group all the approaches related to services, explaining their use and interaction to create business processes and the possibility to add flexibility to such interaction using a component approach. ii) In the second part, Section 2.3, we present the context-awareness in event-driven applications, and then we show how event processing can be used in the business process management. Finally, in Section 2.4 we summarize the ideas presented in this Chapter.

2.2 Services and Component Architectures

Service-oriented architecture (SOA) is a paradigm for the realization and maintenance of business processes that span large distributed systems. It is defined by the Organization for the Advancement of Structured Information Standards (OASIS) as:

> *"A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations"*. [OASIS, 2006].

SOA is based on three major technical concepts: services, interoperability through an enterprise service bus, and loose coupling [Josuttis, 2007].

- A *service* is a piece of self-contained business functionality. The functionality might be simple (storing or retrieving data), or complex (a business process for an online transaction). A service can be seen as an IT representation of some business functionality.
- *Interoperability* refers to the ability of several systems to connect with each other and communicate successfully. An enterprise service bus (ESB) is the infrastructure that enables high interoperability between distributed systems for services. It makes it easier to distribute business processes over multiple systems using different platforms and technologies.
- *Loose coupling* is the concept of reducing system dependencies. It refers to the amount of knowledge that one module has over another one in a system. Since business processes are distributed over multiple back-ends, it is important to minimize the dependencies between different modules. Otherwise, modifications become too risky, and system failures might break the overall system landscape.

When developing applications based on SOA, there are eight design principles we need to consider, according to [Erl, 2007][2]:

- **Standardized service contract.** Services adhere to a communications agreement, as defined collectively by one or more service-description documents.
- **Service loose coupling.** Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.
- **Service abstraction.** Beyond descriptions in the service contract, services hide logic from the outside world.

[2] http://www.soaprinciples.com

- **Service reusability.** Logic is divided into services with the intention of promoting reuse.
- **Service autonomy.** Services have control over the logic they encapsulate.
- **Service granularity.** A design consideration to provide optimal scope and right granular level of the business functionality in a service operation.
- **Service statelessness.** Services minimize resource consumption by deferring the management of state information when necessary.
- **Service discoverability.** Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
- **Service composability.** Services are effective composition participants, regardless of the size and complexity of the composition.

SOA comprises loosely coupled, highly interoperable application services, which inter-operate based on a formal definition independent of the underlying platform and programming language. The interface definition encapsulates the vendor and language-specific implementation. With this, SOA is independent of development technology and the software components become very reusable because the interface is defined in a standards-compliant manner.

2.2.1 Web Services

Most of the SOA community agree that there is only one appropriate way to realize a SOA landscape: with Web Services [Josuttis, 2007]. Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the Web [Team, 2000]. The World Wide Web Consortium (W3C)[3] defines Web Service as follows:

> *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."*

[3] http://www.w3.org/

They rely on three main technologies to support the implementation of SOA: WSDL to describe service interfaces, SOAP to exchange messages, and UDDI to support service discovery.

- *The Web Service Description Language* (WSDL)[4] is an XML-based language that is used to define web service interfaces. It describes two different aspects of a service: first, the functional interface of the service (name and parameters) and second, the technical information about its binding and deployment details (communication protocol and location).
- *The Simple Object Access Protocol* (SOAP)[5] is a standard that defines the Web Services protocol. While HTTP is the low-level protocol, also used by the Internet, SOAP is used to support the exchange of messages between Web Services. Messages are encoded in XML and transported over the network through standard protocols (*e.g.*, HTTP, SMTP).
- *The Universal Description Discovery and Integration standard* (UDDI)[6] is a standard for managing Web Services. It can be seen as a Web Service directory, which allows the publication and discovery of services. It includes information about the service providers and some meta-data about the services themselves (*e.g.*, legal or technical information).

According to IBM, a Web Services architecture requires three fundamental operations: publish, find, and bind [Team, 2000]. To accomplish this, there have to be three main roles in this architecture: a service provider, a service consumer and a service broker. The service providers publish services to a service broker. Then, the service consumers find required services using a service broker and bind to them. This is illustrated in FIG. 2.1.

Another approach to deal with Web Services other than SOAP, is using REST. REST (REpresentational State Transfer) is a resource-oriented software architecture style for building Internet-scale distributed applications [Fielding, 2000]. It defines the principles for encoding, addressing, and accessing a collection of resources using Internet standards. In REST, Web Services are considered as resources, which

[4] http://www.w3.org/TR/wsdl
[5] http://www.w3.org/TR/soap
[6] http://www.oasis-open.org/committees/uddi-spec/

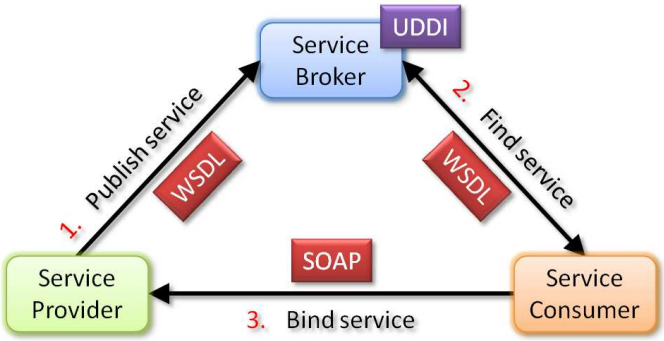


Figure 2.1: Web Service components

can be exposed by means of standard and simple protocols (*e.g.*, HTTP) and a generality of interfaces.

2.2.2 Business Processes and Service Composition

One way to exploit the use of Web Services is by coordinating them to reach a specific goal. This coordination can be expressed in the form of a workflow. Worflows are series of connected steps that represent the flow of operations that need to be executed to achieve a specific task. We may think of them as one primitive building block of organizations, as they serve to create and define business processes, which are defined by the Workflow Management Coalition (WfMC)[7] as:

> *"A set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships."* [WfMC, 1999].

So, in this dissertation we will consider workflows as "*the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*" [WfMC, 1999]. In the workflows, business processes are managed by a "Workflow Management System", using the process instances that were created for its execution. At this time, the activities that compose the business process are also instantiated. The relation between the different parts is better seen on FIG. 2.2.

[7] http://www.wfmc.org/

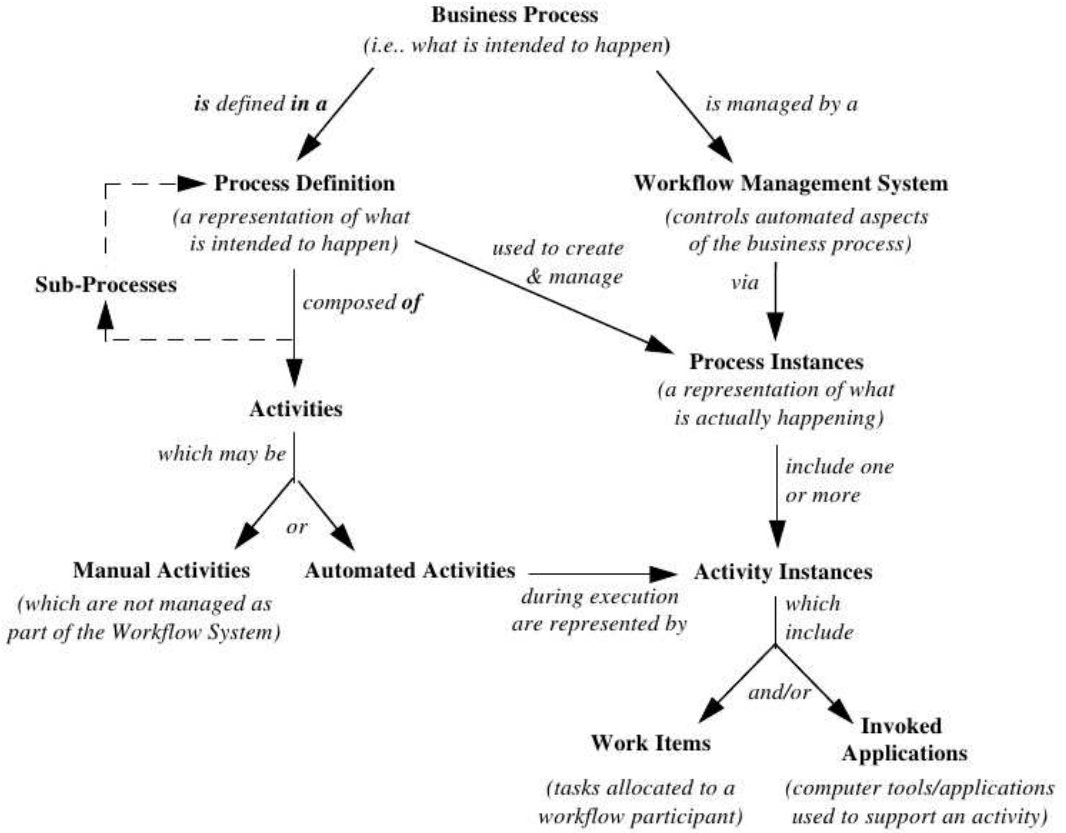


Figure taken from [WfMC, 1999].

Figure 2.2: Business Processes

The coordination of these services to create a business process is known as *Service Composition*, and there are two approaches to it: service orchestration and service choreography [Peltz, 2003]. In service orchestration there is a central element that controls the order in which each of the element of the process interact with each other, while in service choreography there is no central control, and each element of the process is autonomous and know how and when to collaborate with the rest.

An easy way to understand these two approaches is with a dancing performance (*e.g.*, a ballet). The live music is orchestrated by a central person, called the conductor, who controls the pace and the moment in which each instrument

should join. The dance is choreographed, and there is no person controlling the actions of the rest. Each dancer knows their role and execute it in collaboration with the others.

Both approaches have their own advantages, however, the service orchestration approach is the most common and was widely adopted by the industry and even the academy communities. On the other hand, the choreography approach did not get as much support, even though many thought it was the best solution for complex processes [Qiu et al., 2007], and the efforts to enhance it were dropped when the W3C Web Services Choreography Working Group was closed on July 2009[8], which left the orchestration approach as the most viable option for executing business processes, where the Web Service Business Process Execution Language (WS-BPEL) is the *de facto* standard.

2.2.3 WS-BPEL

As the adoption of Web Services began to grow, new standards and specifications began to appear, to help the community to better deal with their new needs. The group of these standards is known as WS–$*$[9], and covers several categories (*e.g.*, security, reliability, messaging, and business process, among others). Our interest is mainly focused on the *Business Processes* category, where the combined efforts of IBM and Microsoft resulted in the creation of the *de facto* standard for orchestrating Web Services, called WS-BPEL (originally known as BPEL4WS) [OASIS, 2007].

As an orchestration language, WS-BPEL (or simply BPEL) specifies the way and order in which each of the Web Services is called, and how the exchange of messages is realized. It defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners, which occurs through the use of Web Service interfaces. A BPEL process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination.

Moreover, a BPEL process is a reusable definition that can be deployed in different ways and in different scenarios, while maintaining a uniform application-level behavior across all of them [OASIS, 2007]. When executing a BPEL process, a central entity is responsible for the invocation and structured combination of the

[8] http://www.w3.org/2002/ws/chor/
[9] http://www.ibm.com/developerworks/webservices/standards/

involved services. Only this central entity is aware of the business process, all other services do not know that they take part in the orchestration.

A BPEL definition is mainly formed by three parts: *activities*, *partner links* and *variables*.

- The *activities* can either be basic or structured. Basic activities are simple actions such as sending or receiving information to/from other services or assigning values to the variables. This kind of activities include for example: `invoke` (which is used to call another service), `assign` (which allows to update the values of variables with new data), and `reply` (which sends the response to the process client). The structured activities are the ones that manage the control flow of the process. Among this kind of activities we can find: `sequence` (executes the activities in a sequential order), `flow` (executes the activities in parallel) and `while` (executes the activities repeatedly until a certain condition is met).

- The *partner links* define the Web Services that are used in the process. Using the attribute `partnerRole`, we can specify the kind of role the partner is playing in the interaction. A business process can also be seen as a Web Service, and in that case, we define its role with the attribute `myRole`. This attribute is also used when dealing with asynchronous scenarios.

- *Variables* provide the means for holding messages that constitute the state of a business process. These messages are often those that have been received from partners or are to be sent to partners. Variables can also hold data that is needed for holding state related to the process and never exchanged with partners. The type of each variable may be a WSDL message type, an XML Schema simple type or an XML Schema element.

To understand how each of these parts are used in the BPEL definition, we will use a simple business process *"Hello World"* example[10], presented in FIG. 2.3. When the business process begins, it receives a value from the caller, which is the name that will be used for the greeting. Then, using the information from the caller, a greeting is created and returned to the client as response.

[10]Example taken from: http://docs.jboss.com/jbpm/bpel/v1.1/userguide/tutorial.hello.html

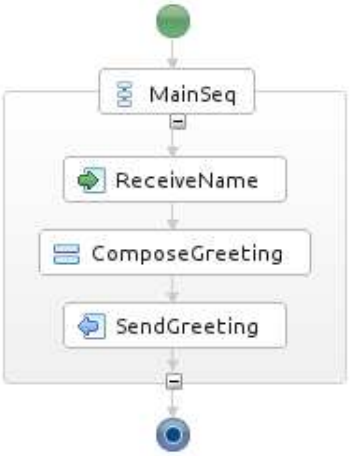


Figure 2.3: BPEL example

In FIG. 2.4 we can see the BPEL code used to create this process. We can see the definition of the `partnerLink` in lines 7-11. In lines 13-16, two variables are defined: one to receive the information (name) from the client, and another one to store the response that will be sent back. The process definition begins in line 18, with a sequence, and the three activities (receive, assign and reply) are defined in their order of execution. During the assign activity, we can either copy the value of a variable into another, assign a fixed value to a variable, or like in this case, in line 25, use an XPath expression to manipulate the information that is going to be assigned.

Besides the definition of the business process in the BPEL file, we also need to provide the definitions of the *partnerLinks* using a WSDL file, which we present in FIG. 2.5. In this part, we define the incoming and outgoing messages that will be exchanged, which may include several parts (or variables), as well as the interactions that will occur according to the `portType` that will be used. Finally, we define the partnerLink assigning it a specific role and portType.

The BPEL standard has been widely accepted and adopted in the industrial and academic communities, and there has been a lot of research made around it in many domains, like security [Zaplata et al., 2009, Majernik et al., 2011, Nassar et al., 2009], quality of service [Mukherjee et al., 2008, Christos et al., 2009, Baligand et al., 2007] or even modeling [Lin et al., 2008, Juhnke et al., 2010].

```
<process name="HelloWorld"
  targetNamespace="http://www.example.com/hello"
  xmlns:tns="http://www.example.com/hello"
  xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>
    <partnerLink name="caller"
                 partnerLinkType="tns:Greeter-Caller"
                 myRole="Greeter" />
  </partnerLinks>

  <variables>
    <variable name="request" messageType="tns:nameMessage" />
    <variable name="response" messageType="tns:greetingMessage" />
  </variables>

  <sequence name="MainSeq">
    <receive name="ReceiveName" operation="sayHello"
             partnerLink="caller" portType="tns:Greeter"
             variable="request" createInstance="yes" />
    <assign name="ComposeGreeting">
      <copy>
        <from
expression="concat('Hey ', bpel:getVariableData('request', 'name'), '!')"
        />
        <to variable="response" part="greeting" />
      </copy>
    </assign>
    <reply name="SendGreeting" operation="sayHello"
           partnerLink="caller" portType="tns:Greeter"
           variable="response" />
  </sequence>

</process>
```

Figure 2.4: BPEL source code

However, it still has some limitations, like its lack of flexibility to be adapted at run-time, without having to redeploy the whole process every time a change is made. To solve this problem, we need to incorporate into BPEL a mechanism that allows us to add the capacity of dynamic adaptation. In Chapter 3 we will present some of the different approaches that have been used in this regard. One way to add flexibility to the service composition is by using a service component approach, as we will present in the following section.

```
<definitions targetNamespace="http://www.example.com/hello"
  xmlns:tns="http://www.example.com/hello"
  xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="nameMessage">
    <part name="name" type="xsd:string" />
  </message>

  <message name="greetingMessage">
    <part name="greeting" type="xsd:string" />
  </message>

  <portType name="Greeter">
    <operation name="sayHello">
      <input message="tns:nameMessage" />
      <output message="tns:greetingMessage" />
    </operation>
  </portType>

  <plt:partnerLinkType name="Greeter-Caller">
    <plt:role name="Greeter">
      <plt:portType name="tns:Greeter" />
    </plt:role>
  </plt:partnerLinkType>

</definitions>
```

Figure 2.5: WSDL source code

2.2.4 Service Component Architecture

A software component is an architectural entity that encapsulates a subset of the system's functionality and/or data, restricts access to that subset via an explicitly defined interface, and has explicitly defined dependencies on its required execution context [Taylor et al., 2007]. Component-Based Software Engineering (CBSE) allows programs to be constructed from pre-built software components, which are reusable, self-contained blocks of computer code. These components have to follow certain predefined standards including interface, connections, versioning, and deployment [Wang and Qian, 2005]. Component-oriented programming enables the development of software by assembling independent components into a software architecture.

SCA (Service Component Architecture) is a set of specifications for build-

ing distributed applications and systems using the principles of SOA and CBSE [Beisiegel et al., 2007]. Components are at the core of the SCA Model, and the SCA specifications define how to create and combine those components into complete applications [Chappell, 2007]. A component usually implements some business logic, which is exposed as one or more services. At the same time, a component might also rely on services provided by other components (inside or outside if its domain), in which case it can rely on the use of references. Finally, a component can also define one or more properties, each of which contains a value that can be read when the component is instantiated.

SCA is neutral with respect to programming languages, and can be implemented in any language that supports it. For example, components might be built with Java or other languages using SCA-defined programming models, or even built using other technologies, such as BPEL. It defines a common assembly mechanism to specify how those components are combined into applications.

The SCA graphical representation is basically formed by six main parts: *components*, *composites*, *services*, *references*, *properties*, and *wires*. All of these parts are exemplified in FIG. 2.6, and explained below.

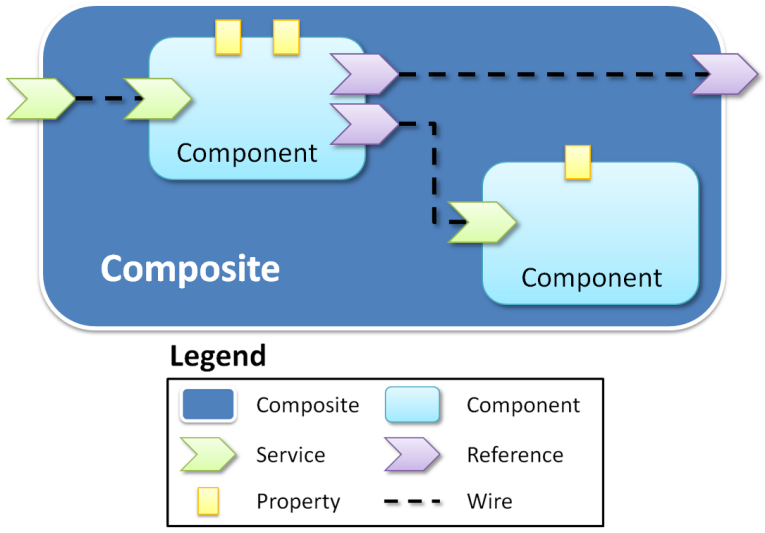


Figure 2.6: SCA diagram

- As mentioned before, a *component* is the basic element of business function in an SCA assembly. When combined with others, they can interact to create

complete business solutions. Basically, a *component* is a configured instance of an implementation, that provides and consumes services. This implementation is the code that actually provides the component's functions, such as a Java class or a BPEL process.

- A *composite* is used to assemble SCA elements in logical groupings. It represents a way to combine components into a larger structure. An SCA composite contains a set of components, services, references and the wires that interconnect them. A composite may form component implementations in higher-level composites and its content may be used within another composite through inclusion, resulting in all of its contents made available for use within the including composite.

- A *service* represents the way to access the functionality of a component. It exposes the catalog of operations provided by the component, using an interface, and it can be accessed within the same domain, or exposed publicly. The services of a component can be promoted by a composite, exposing the component's service as its own.

- A *reference* represents a functionality that is required by its component. Using an interface, references can be bound to its corresponding services. In the same way as services, the references of a component can be promoted by a composite.

- A *property* allows for the configuration of an implementation with externally set data values. All the properties are typed, and a default value can be defined by an implementation. Properties are configured with values in the components that use the implementation.

- A *wire* is an abstract representation of the relationship between a reference and some service. The kind of communication provided by a wire can vary, since it depends on several factors (*e.g.,*. the specific run-time and specified bindings).

There are several different implementations of the SCA specifications, either from commercial providers like IBM and Oracle, or from open source efforts like Tuscany, Fabric3 and FraSCAti. The list of this implementations can be found on

the Open SOA website[11]. Since the goal of our project has to do with dynamic adaptation and reconfiguration of services, we will focus on the FrasSCAti platform, as dynamic reconfiguration is one of the added values of this platform that the others do not provide [Seinturier et al., 2009, Seinturier et al., 2012].

FraSCAti is a platform for developing SCA based distributed systems, which brings run-time adaptation and manageability properties to SCA applications and their supporting platform. Its advantage is that it introduces reflective capabilities to the SCA programming model, that allow dynamic introspection and reconfiguration. These features open new perspectives for bringing agility to SOA and for the run-time management of SCA applications.

The platform itself is built as an SCA application (its different subsystems are implemented as SCA components), and provides an homogeneous view of a middleware software stack where the platform, the non-functional services, and the applications are uniformly designed and implemented with the same component-based and service-oriented paradigm. With FraSCAti, the structure of an SCA application can be discovered at run-time using introspection, and modified dynamically to add new services, or even reconfigured to take into account new operating conditions.

2.3 Event-driven and Context-aware Applications

We have seen how services help to improve the execution of business processes. However, service-oriented architecture does not address all the capabilities needed to respond to the dynamicity of today's environments, where the context is constantly changing and we need to be able to monitor those changes in order to be able to respond in time and form. In this respect, this section introduces the notions of context and how the information about it can be obtained and processed using an event-driven approach.

2.3.1 Context definition

Context-awareness refers to an application's ability to react to changes in the environment and use context information during its execution. Context-aware com-

[11] http://www.osoa.org/display/Main/Implementation+Examples+and+Tools

puting was first discussed in 1994 to be software that "adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time" [Schilit and Theimer, 1994]. We can say that an application is context-aware if it uses context to, for example, provide relevant information and/or services to its user, where the pertinence of such information/services depends on the user's current task. In this case, context can be defined as:

> *"Any information that can be used to characterize the situation of entities* (i.e., *whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects."* [Dey et al., 2001].

Event-driven applications are based on the Event-Driven Architecture (EDA), that is an architectural style in which one or more components in a software system execute in response to receiving one or more event notifications. The previous context definition also applies when working with context in event-driven applications. However, there are certain notions that are only related to events, in which case an interesting context definition can be the one provided by Etzion and Niblett:

> *"A context is a named specification of conditions that groups event instances so that they can be processed in a related way. It assigns each event instance to one or more context partitions. A context may have one or more context dimensions and can give rise to one or more context partitions."* [Etzion and Niblett, 2010].

Context information can come from several different kinds of sources, *e.g.*, sensors, web services, instant-messaging systems, and relatively static repositories such as databases and calendars. In event-driven applications, this context information can be interpreted using one or more of the following domains [Etzion and Niblett, 2010]:

- *Temporal context*, which consists of one or more time intervals (also known as windows), that can possibly be overlapping. Each of these time intervals correspond to a context partition, which contains events that happen during that interval.

- *Spatial context*, where the different event instances are grouped according to their geospatial characteristics. This type of context assumes that the event contains an attribute that assigns a location to the event.
- *Segmentation-oriented context*, that assigns the events to specific context partitions, based on the values of one or more event attributes, either by using the value of these attributes to directly assign a partition, or by using predicate expressions to define the membership to a context partition.
- *State-oriented context*, which is a context controlled by an external entity, and the decision of whether an event is to be included in the partition is based on the state of such external entity at the time when the event occurs or is detected. This dimension differs from the others in that the context is determined by the state of some entity that is external to the event processing system, and that has a only a single partition.

These contexts are often not used alone, but in combinations of two or more different types, which are called *composite contexts*. The contexts composing a composite context are called *members*. When applying composite contexts to series of events, we need to define a priority for each of the members, since the order in which we apply each of the member contexts can change the final result. For instance, if we have a composite context containing temporal and segmentation-oriented members, if our segmentation is done by any attribute other than its temporal identifier, the order in which they are applied will result in a different outcome.

2.3.2 Event Processing

In order to use all the context information, we first need to be able to process the streams of events that are coming from the different sources. This is called *Event Processing*. Event processing is an emerging area which refers to an approach to software systems that is based on the idea of events, and that includes specific logic to filter, transform, or detect patterns in events as they occur. Luckham defined the relation between event-driven systems and event processing as follows:

> *"Event-driven systems provide automation to allow the events to be interpreted and correlated, and support the aim of delivering a timely response. Event*

processing is a set of techniques and tools that help us understand and control event-driven systems." [Luckham, 2002].

There are three main roles in an event processing: *event producers*, *event consumers*, and *event processing agents*.

- An *event producer* is the entity that introduces the event to the system, and can be represented in a variety of ways of either physical or virtual entities (*e.g.*, hardware sensors, software applications or business processes).
- *Event consumers* are the ones that receive the events coming from the system, that can as well be represented in a variety of ways (*e.g.*, hardware actuators, data stores, business processes). An entity can be at the same time an *event producer* and an *event consumer*.
- The *event processing agent* is a software module that stands in the middle of the *event producers* and the *event consumers*. It analyzes and processes the event, and can use its information to filter the event, detect a pattern or even transform it and create new events.

Given this, events can be of two types: *raw events*, and *derived events*. A *raw event* is the one that is introduced into the system by an event producer, and its definition relates only to its source and not to its structure (*e.g., "CPU charge: 80%"*). A *derived event* is the one that is generated as a result of event processing that takes place inside an event processing system (*e.g., "The system is overloading"*). The type of event is relative to the system where it is being considered, thus a derived event that is sent to a second event processing system will be perceived by it as a raw event.

To specify the way in which these events are processed by the event processing agent, we need an event processing language. Even though the logic of the event processing can be implemented using standard programming languages (*e.g.*, Java or C#) within our systems, just as we use the Structured Query Language (SQL) to send requests to a database instead of reprogramming the whole database logic, we should as well profit from the several advantages provided when using the dedicated event processing languages.

As can be seen in TAB. 2.1, there are at least three big categories of event processing languages (stream-oriented, rule-oriented and imperative), and several implementations of each.

Language style		Product name	Vendor	Open source
Stream-oriented		Aleri	Aleri	No
		CCL	Aleri/Coral8	No
		Esper	EsperTech	Yes
		CQL	Open ESB IEP	Yes
		Oracle CEP	Oracle	No
		RTM Analyzer	Realtime Monitoring	No
		SPADE	IBM	No
		StreamInsight	Microsoft	No
		StreamSQL EventFlow	StreamBase	No
Rule–oriented	*ECA Rules*	Amit	IBM	No
		AutoPilot M6	Nastel	No
		Reakt	RuleCore	No
		RulePoint	Informatica	No
		SENACTIVE InTime	UC4 / SENACTIVE	No
		StarRules	Starview Technology	No
		Vantify	WestGlobal	No
		WebSphere Business Events	IBM	No
	Inference rules	DROOLS Fusion	Jboss (RedHat)	Yes
		TIBCO BusinessEvents	TIBCO	No
	Logic Programming	Etalis	Open source project	Yes
		Prova	Open source project	Yes
Imperative		MonitorScript	Progress Software	No
		Tivoli/Netcool IPL	IBM	No

Table based on the one provided in [Etzion and Niblett, 2010].

Table 2.1: Event processing language classification

- The **stream-oriented** engines are concerned with the latest data and keep very little history in memory and process the data asynchronously. Most of the stream-oriented languages are greatly inspired by the Structured Query Language (SQL) used for managing data in relational databases.

- **Rule-oriented** languages aim to combine event processing for real-time event detection and reaction rules for declarative representation and intelligent reaction [Paschke and Kozlenkov, 2009]. They often use event notification and messaging systems, such as Enterprise Service Bus (ESB), to facilitate the communication of events in a distributed environment.
- Finally, the **imperative languages** are the ones where the logic is coded in syntax similar to the one used on the C or Java programming languages.

Since a standard has not yet been defined as of how to deal with the event processing, even the languages that are in the same category can be very different from each other. So, when choosing an event processing technology, we are basically marrying the product family of our choice for a while.

2.3.3 Using Event Processing in BPM

Business Process Management (BPM) is a collection of methods, policies, metrics, management practices, and tools used to design, run, and manage systems that support a company's business processes [Chandy and Schulte, 2010].

Business processes always involve business events in a general sense. Conventional BPM engines control the flow of the process by evaluating conditions referring to the events that are generated by the application software in which the business process is running. In this way, BPM engines have a restricted knowledge of their environment, and they can take decisions based only on the information provided by the process, in contrast to real context-awareness, where the systems can benefit from a wide variety of information sources that provide it with the information to better understand its environment and execute accordingly.

To extend this awareness, the BPM engine can be complemented with an Event Processing (EP) engine. The EP engine can get information about the events that happen outside the business process from sources such as sensors, the Web, or other application systems. At the same time, the EP engine can get information about the events that are happening inside the business process from the BPM software, giving the system a complete context-awareness. From these base events, it can create derived events, that are then forwarded to the BPM software to enable sophisticated, context-dependent, situation-aware decisions. This collaboration between BPM and EP was well exemplified by Mani Chandy in FIG. 2.7.

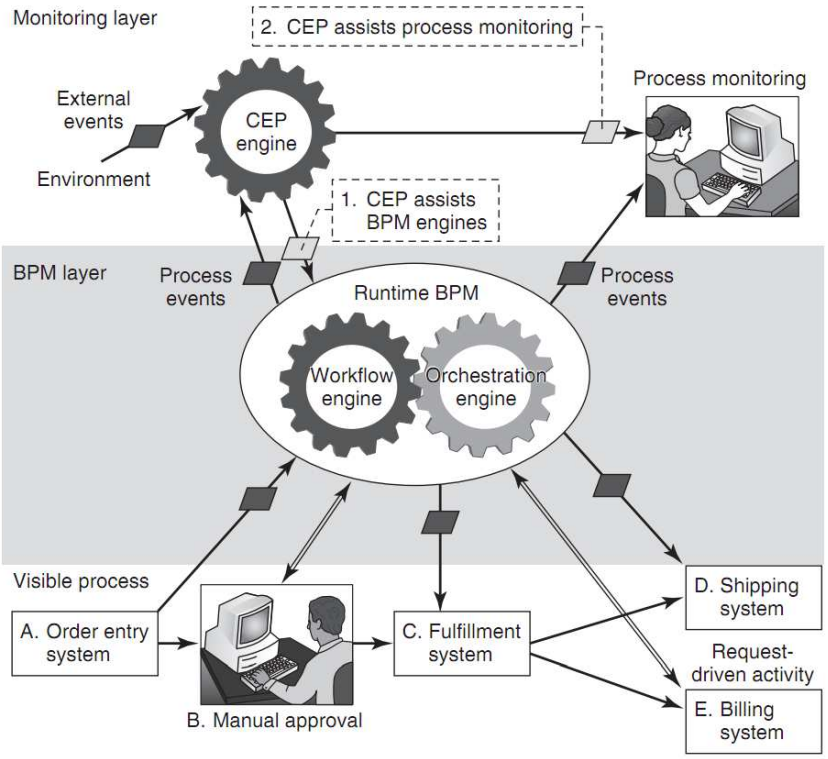


Figure taken from [Chandy and Schulte, 2010].

Figure 2.7: Using Event Processing with BPM

The future of this collaboration is going towards a new approach called "Event-Driven Business Process Management" (EDBPM), which is an enhancement of BPM by a combination of several concepts (old and new), such as Service Oriented Architecture, Event Driven Architecture, Software as a Service, Business Activity Monitoring and Complex Event Processing (CEP) [von Ammon et al., 2009]. However, while this bundle of different technologies looks to be moving in the right direction, the natural inflexibility of orchestration engines mixed with the wide diversity of CEP approaches, create a gap between the desired features and the provided functionality. This will lead to the evolution of current systems towards more dynamic and flexible orchestration technologies, which can be constantly adapted to new situations presented by their surroundings, creating a stronger bond between business processes and context information.

2.4 Summary

In this chapter we have introduced some of the concepts that we will use throughout the dissertation, which we divided into two main groups: Service-oriented applications and Context-awareness using events. We explained how we used services to create business processes and defined their execution using BPEL. We presented the Service-Component Architecture as a solution to provide flexibility to service composition, specially thanks to the dynamic reconfiguration capabilities provided by FraSCAti.

We have also shown how business processes could benefit from the use of context information and how we could integrate this information to the process using event processing. We finally showed that there is a trend towards coupling BPM with Event Processing techniques, to create Event-Driven Business Process Management. However, since the engines executing the business processes still inherit the rigid nature of context-unawareness, there is still a long way to go before this coupling is really exploited by out-of-the-box solutions.

In the next chapter we will present some of the works related to business process adaptation, that are trying precisely to add the needed flexibility to the process using several approaches. We will evaluate them and try to find a better approach in order to create a fully dynamically adaptable context-aware business process.

Chapter 3

Business Processes and Adaptation

"Complexity is a sign of technical immaturity. Simplicity of use is the real sign of a well design product whether it is an ATM or a Patriot missile."
- Daniel T. Ling

Contents

3.1 Introduction

As the use of business processes continues to grow, the environment in which they are being executed is becoming more and more dynamic, with constant changes affecting the performance and overall outcome of the business processes. However, current implementations of business process managers provide only static executions, which contrasts with the strong need to respond to those changing environments by updating our business processes accordingly.

Dynamic business process adaptation allows to add flexibility to the processes, by providing the tools to adapt the execution of the business processes to the new conditions. Generally speaking, adaptations can be i) static or dynamic, ii) manual or automatic, and iii) proactive or retroactive [Courbis and Finkelstein, 2005]. Static adaptations are carried out through modifications in the source code, while dynamic ones modify the software characteristics at run-time. Manual adaptations require direct human intervention in the system, while automatic ones can be performed by the system itself (self-adaptive systems) when a certain condition is reached. Finally, proactive adaptations occur before a specific event, while reactive adaptations happen after it, as a consequence.

In this dissertation we will be focusing on dynamic adaptations of business processes, and so, in this Chapter we evaluate different approaches for it. Dynamic adaptation has been a widely studied topic [Cheng et al., 2009], and not only on the business process domain. For instance, the MUSIC middleware [Rouvoy et al., 2009] is defined to support component assembly self–adaptation. Event-based AOP (EAOP) is a framework that intends to express aspects in terms of events that arrive during execution [Douence et al., 2001]. They even detect sequences of events, and relate them using *event patterns* at run-time.

In the literature we can find that there are mainly two kinds of business process adaptations: vertical adaptation and horizontal adaptation [Papazoglou, 2008]. Vertical adaptation refers to the changes made to the binding of the services to their respective implementations (or providers), without affecting the structure of the process (*i.e.*, order in which the activities are executed), while horizontal adaptation refers to the capability of modifying (adding, changing or removing) fragments of the structure of the process.

In the following sections we present several works that offer a solution to the lack of flexibility on business processes with different approaches. We compare

these approaches using different criteria related to the type of adaptation that they use and the mechanisms to achieve it. In this context, we segment our comparison in two groups: vertical adaptation approaches and horizontal ones, since their goals are slightly different and some of the criteria can only be used for one of the segments. Finally, we also present some works related to undoing adaptations. This is usually considered as a trivial task, however it is an important complement of dynamic adaptations and when done without the proper care, it can lead to inconsistent systems.

Structure of the Chapter

The rest of this chapter is organized as follows: We begin by presenting some of the existing work related to vertical adaptations in Section 3.2. Then, in Section 3.3, we present some of the work that applies horizontal adaptations to the processes. In both cases we evaluate the approaches using different criteria that is also explained within the sections. In Section 3.4 we show some of the work that has been done for undoing adaptations in a general context (*i.e.,* not only business process adaptations). In Section 3.5 we retake the challenges presented in Chapter 1, and position them according to the State of the Art. Finally, we conclude in Section 3.6 with a summary of the ideas presented in this Chapter.

3.2 Vertical business process adaptation

Vertical business process adaptation refers to the changes done at a service-level (*i.e.,* service rebinding), that do not affect the structure or sequence of execution of the process. Until now, this is where most of the work regarding business process adaptation has been focusing. This is mainly because one of the major concerns about the adoption of Web Services as the channel to execute business processes is the Quality of Service (QoS) and the need from the service providers to respect certain Service Level Agreements (SLA). Most of these works have as a goal to improve the QoS and prevent SLA violations, and to do that they rely on analyzing several service providers and selecting the best option for the required metrics.

In business processes, the objective of vertical adaptations is not to modify the behavior of an existing process, but to give some assurance that the process will be successfully executed, and that the service provided by the partners of the process

will comply to the predefined criteria. Now we will present some of the works that use this approach as the means to add certain flexibility to the business process execution, which we will then evaluate using different points of comparison, and finally discuss their advantages and disadvantages at the end of this Section.

3.2.1 Vertical adaptation approaches

In [Canfora et al., 2008], the authors present a framework for binding and re-binding of composite web services. They integrate QoS monitoring to the services of the business processes, and use time, price, availability and reliability (as well as custom attributes) as a way to measure it. The business process is first created as an abstract workflow, without any bound services. Once the workflow is defined, the services are bound to a first set of providers, taken from a pool, always considering the QoS metrics. If the Service Level Agreement is violated during the execution of the process, or if the QoS deteriorates indicating a high probability of this happening, the system adapts the process and re-binds the services to new providers from the pool that respect the QoS.

An approach to rebind BPEL processes is presented in [Strunk et al., 2009]. They provide the user with an interface to define alternative services at design time, which will be used to adapt the process. Their solution is composed by three parts: a proxy layer, a monitoring component and a rebinding component. The proxy layer receives the requests from and responds to the BPEL engine, allowing a flexible point to change the service provider when needed. The monitoring component watches for events and triggers the rebinding component when SLA violations are detected. The rebinding component replaces a failing service with a equivalent one, from a pool of services that was previously specified at design time.

In [Bastida et al., 2008], the authors introduce an approach for context-aware service composition using a methodology of six steps to define an executable model composed of several services. Their approach is based on the notions of Product Line Engineering for variability management [Pohl et al., 2005]. The final process has a set of variants chosen for several variation points, which will connect to new services based on the interpretation of some ECA (Event-Condition-Action) rules. The authors express the adaptation conditions using their own language, which associates a programmed reconfiguration action to a property of the context information. During execution, when an event indicating a reconfiguration condition arrives, the variation point is bound to a new service.

An aspect–oriented solution, using the Spring .NET framework, is presented in [Rahman et al., 2008]. They use a contract-based approach to assign a web service to each instance of an execution call. Using ECA rules, they create a rule-based contract between the participating partners. To achieve adaptation of the process they can change the contract of the ECA rules at run-time and assign a new web service for the call. They can also adapt an existing implementation of a web service by using aspects to weave the new behavior.

The authors in [Lins et al., 2007] propose to deal with process adaptation by adding a web service repository that handles the web services to invoke in each case. Whenever an invocation of a web service is done, the call is intercepted and the repository is checked for changes in the process definition, before the invocation of a web service. If there have been some changes, then it examines the available web services in the repository and chooses the one that best suits the criteria, otherwise the invocation is executed as usual.

In [Colombo et al., 2006], the authors present a platform called SCENE, which integrates a BPEL execution engine and ECA rules using Drools. They propose a composition language through which they describe service compositions in terms of two distinct parts: a process part, described using BPEL, that defines the main business logic of the composition, and a declarative part, described using ECA rules. These rules are checked at run-time and are used to realize the correct bindings between the BPEL engine and the services, and they are specified in terms of events that are generated by activities specified in the BPEL definition. The events are related to the business logic and are previously defined using Drools. They are monitored using the variables from the activities. The activities that can be adapted, are at first bound to a proxy, which depending on the state of the process then forwards the request to the corresponding service provider.

The authors of [Cruz Torres et al., 2010] present a framework called CASAS (Composable, Adaptive, Service, Agent System), which offers a solution based on the Multi-Agent System (MAS) model. They add another level of abstraction to the composite service model, called the organization layer, that explicitly represents the interactions between all the services participating in the composition, the adaptation constraints and the expected behavior of the composition. It relies on an Enterprise Service Bus (ESB) for communication between the BPEL engine, the web-services and the agents, acting as a type of evolved proxy. Using the agents, the system is responsible for dynamically providing the best partner services to the

workflow instances. This selection is done in compliance to the SLA of each partner web-service.

3.2.2 Comparison criteria for vertical approaches

To compare the different approaches of vertical adaptation presented in this document, we will use the following criteria:

- **Change service provider**. This criteria refers to the capability of the solution to dynamically change the binding of the service from one provider to another one, without the need to redeploy the process.
- **Change activity behavior**. This criteria refers to the capability of the solution to change, not only the service provider, but also the type of service that would be called, thus changing the behavior of the activity.
- **Context monitoring**. This criteria refers to the capability of the solution to monitor the conditions under which it would adapt the process. In this case we specify what kind of approach they are using to monitor the context and any additional complement to do it.
- **Context scope**. This criteria refers to the limits of the information sources to be considered for the context monitoring. It is usually either focused only on the service performance, or more openly on the whole business process behavior, or without restrictions considering any possible source of context information.
- **Automatic adaptation**. This criteria refers to the capability of the solution to react to the adaptation conditions and automatically make the necessary changes to the business process, without any human interaction.
- **Proactive adaptation**. This criteria refers to timing of the adaptation. As mentioned earlier, adaptations can either be proactive or reactive. If the adaptation is only triggered in response to a specific event (*e.g.,* an SLA violation), then it is only a reactive adaptation. However, if it occurs beforehand, while a specific situation is probable to arrive, but has not yet happened (*e.g.,* a decrease in the QoS, without violating any SLA yet), then the adaptation to avoid that situation is considered proactive.

- **Adaptation mechanism**. This criteria refers to the specific approach used by the solution to achieve the adaptation of the business process. All of these mechanisms are used to add flexibility to the execution of business processes, however every approach has its own characteristics and allows different kinds of manipulations of the process. Some, like the *proxy* mechanism, allow only to intercept and redirect service calls for adapting the service provider of an activity, while others, like *aspects*, could allow to modify the behavior of an activity or even alter the structure of the whole process.

3.2.3 Discussion of vertical adaptation

In this Section we have presented some of the work that has been done in the vertical adaptation of business processes. As can be seen in TAB. 3.1, the most common adaptation mechanism in this kind of approaches is the use of a proxy. Since the idea of a vertical adaptation is to be able change the provider of the service, a good way to do it is to intercept the call and then apply the different techniques to forward the call to the best possible provider.

The solution proposed by [Rahman et al., 2008], is in the limit between vertical and horizontal adaptation. It is mainly focused on a vertical approach, however, thanks to the use of aspects, they are able to change the implementation of an activity, which could be considered as a horizontal adaptation, but since it is not modifying the order of execution nor adding any additional activities, we decided to consider it among the vertical approaches.

Another thing to notice is that even though the scope of the adaptation remains at the service level, the context scope in some of the solutions is not only limited to the service, but opens up to obtain information from different sources. Specially in the case of [Colombo et al., 2006] and [Bastida et al., 2008], where thanks to the use of an event-driven approach for the context monitoring, their context scope is completely open. An open context scope will allow the adaptation to benefit from a real context-awareness, that covers all the possible information that could be important to the execution of the process. Nevertheless, they only use basic ECA rules, which still lack some discernment about the information that is received, compared to other event approaches like Complex Event Processing (CEP).

The effort to broaden the scope of the context is a good sign of how it is becoming important for business processes to consider context information that is

Reference	Change service provider	Change activity behavior	Context monitoring		Context scope	Automatic adaptation	Proactive adaptation	Adaptation mechanism
Bastida et al., 2008	Yes	No	ECA rules	Events	General	Yes	Yes	Proxy
Canfora et al., 2008	Yes	No	Service monitoring	–	Service context	Yes	Yes	Proxy
Colombo et al., 2006	Yes	No	ECA rules	Events	General	Yes	Yes	Proxy
Cruz Torres et al., 2010	Yes	No	Rules	Agents	Service context	Yes	No	Proxy
Lins et al., 2007	Yes	No	Policy rules	Constraints	Service context	Yes	No	Proxy
Rahman et al., 2008	Yes	Yes	ECA rules	Events	Business process	Yes	No	Aspects
Strunk et al., 2009	Yes	No	Service monitoring	Events	Service context	Yes	No	Proxy

Table 3.1: Summary of vertical approaches

coming from other sources besides its own. However, while these approaches all seem to solve a specific concern, which is mainly related to QoS, the actual need for adaptation in business processes goes beyond that. The dynamicity of current environments requires the business processes to be able to change not only from one service provider to another with better QoS, but also to be able to modify the behavior of the process to better respond to the new context in which it is being executed. In the next Section we will see how horizontal adaptations can help to solve that.

Advantages of vertical adaptation. One of the advantages of using a vertical adaptation is that it is a less invasive approach, meaning that it doesn't need to modify the business process execution engine. This allows it to be used with most of the standard solutions available in the market. Another advantage is that, since it doesn't need to modify the structure of the process, these adaptations are usually faster, and with the use of a dynamic pool of service providers, we can continuously improve our choices.

Disadvantages of vertical adaptation. The main disadvantage of this kind of adaptation is that it is limited in the flexibility that it can offer, as it is focused as a solution to maintain the QoS of the process and not really to respond to new needs. Although it is thought as a solution at a service level, the flexibility needed by current business processes to respond to a changing environment goes beyond that, to the point where the changes in the execution environment can be compensated with adaptations in the way the process is executed.

3.3 Horizontal business process adaptation

Horizontal business process adaptation refers to the modifications made to the structure of the business process, which include the addition and removal of activities from the process to change its behavior. The static nature of business process definitions is now incompatible with the extreme dynamicity of today's environments, and the lack of flexibility hinders the productivity of our processes. As we presented in the previous Section, several efforts have been made to overcome this problem. Until now, most of the approaches focus on vertical adaptation, however, the need to be able to modify our business process behavior to face the fast and

constant changes in the execution environment is growing, and so we have seen more works starting to provide the means to create horizontal adaptations.

Horizontal adaptations allow a broader manipulation of the business process, compared to vertical approaches, which are limited at service-level modifications. The goal of these kinds of adaptation is to allow the business processes to respond to new needs by modifying its behavior without the need to redeploy them. We will now present some of the works that use this approach to respond to the need of flexibility of business processes, and then we will evaluate them using a similar criteria as for the vertical approaches, with the addition of a few comparison points related only to horizontal approaches, and at the end of the Section we will discuss the advantages and disadvantages of this approach.

3.3.1 Horizontal adaptation approaches

In [Sánchez and Villalobos, 2008], the authors use an aspect-oriented approach, focused on separation of concerns and instrumentation. They introduce the `executable models`, which are used to represent the cross-cutting concerns. They use `open objects`, which are representations of the state of the elements in the model, to monitor the invocation of services and adapt the process by weaving the interaction with other models before (activation) and after (deactivation) the call to the service. Their goal is to create workflow applications as executable models which can then be synchronized using method calls and event passing.

An adaptation of the BPEL language called VxBPEL is presented in [Koning et al., 2009]. The authors insist on the need of flexibility and variability in the service-based systems and the lack of them when deploying BPEL processes. They extend the BPEL language to add new elements like `Variation Points`, which are the places where the process can be adapted and `Variants`, which define the alternative steps of the process that can be used. VxBPEL also accepts new `Variants` to be added at run-time, allowing the systems to be adapted without redeploying the process. To support relations between variation points, VxBPEL enables to group related variation points into higher-level variation points, called "configurable variation points", which are embedded in business process definitions.

Another extension of BPEL, using aspects, is introduced in [Charfi et al., 2009]. The authors present a plug-in based architecture for self-adaptive processes that

uses AO4BPEL. Their proposal is to have different plug-ins with a well-defined objective. Each plug-in has two types of aspects: the `monitoring aspects` that check the system to observe when an adaptation is needed and the `adaptation aspects` that handle the situations detected by the monitoring aspects. Whenever the conditions of a `monitoring aspect` are met, it uses AO4BPEL to weave the `adaptation aspects` into the process at run-time. Monitoring aspects can be hot-deployed to their BPEL engine, allowing them to add or change the adaptation conditions at run-time.

The authors in [Leitner et al., 2010], propose a framework called PREvent, which is a system that integrates event-based monitoring, prediction of SLA violations using machine learning techniques, and automated run-time prevention of those violations by triggering adaptation actions in service compositions. Their framework is mainly composed by three parts: a Composition Monitor, an SLO Predictor, and a Composition Adaptor. The Composition Monitor is responsible for monitoring the run-time data, while the prediction of violations are handled by the SLO Predictor. It uses learning techniques to identify the services that can cause SLA violations in the future. Finally, the Composition Adaptor is responsible for identifying and applying adaptation actions. They define the events to be monitored using the Esper Query Language (EQL) and use the results of those definitions to trigger an adaptation in the business process. Their solution is based on the VRESCO run-time environment, which provides the bases used for monitoring and adaptation. They describe their adaptation capabilities as *limited*, concerning the adding and removal of activities, since they cannot modify structured activities and can only add or remove a limited number of simple activities.

The ALLOW framework is presented in [Marconi et al., 2009]. Here, the authors provide adaptation through the use of Adaptable Pervasive Flows. ALLOW's flows are capable to check deviations on the behavior of the entity they are attached to, as well as problems in the execution context, and to trigger adaptation. The flows are modeled in a way that they are logically attached to physical entities, and can be used to model workflows that are related to specific objects. They present a new language called Adaptable Pervasive Flow Language (APFL), which is an extension to BPEL, that takes the context into account in order to adapt the execution of the business processes, providing alternative flows.

In [Zhai et al., 2008] the authors present a reflective framework to improve the adaptability of BPEL-based web service composition. They define a meta-model to build the self-representation of the web services composition. This meta-model

will be modified to adapt to the changing environment, and then, the reflection mechanism utilized in the framework will adjust the web services composition automatically. Using this approach they are able to add and remove activities and link from the original process in the meta-model and reflect the changes to the running process.

The authors in [Xiao et al., 2011] present a constraint-based framework for dynamic business process adaptation. Their approach uses `fragments of processes`, which are isolated compositions of activities that are designed to accomplish a specific task, and may contain any kind of activities (*e.g.,* invocations, loops, etc.) and even introduce its own variables. They use these fragments to complement the business process in pre-designated parts called `variable points`. Using constraints, they can determine which of the fragments can better accomplish the task and then use it to compose the final business process. If the constraints change, their system can then consider new fragments and substitute the older ones, however this changes will only be visible to the new instances of the process.

In [Geebelen et al., 2010], the authors present a framework based on the Model-View-Controller (MVC) pattern. In their approach, the workflow process is designed as a template, and the tasks are specified on an abstract level. Those tasks are modeled as aspects, and their implementation is stored in a library, where they are selected according to policies of the adaptation logic. The library contains aspects of different activities that can be modularized as a specific task. The adaptation policies are properties or parameter values of the executing process. They can rollback an adaptation by restoring the process to a previous state.

3.3.2 Comparison criteria for horizontal approaches

Besides the criteria used to compare the vertical approaches, for the horizontal approaches we will added three more characteristics:

- **Add activity**. This criteria refers to the capability of the solution to dynamically add new activities to the business process, without the need to redeploy it. This criteria will allow the behavior of the process to be modified at runtime, allowing a better response for special situations.

- **Delete activity**. This criteria refers to the capability of the solution to dynamically delete one or more activities from the business process without the need to redeploy it. Just like when adding activities, deleting them modifies the behavior of the process, however, it is easier to leave the business process in an inconsistent state by removing activities than by adding them, so this should usually be done carefully.

- **Adaptation undoing**. This criteria refers to the capability of the solution to undo the changes done to the business process when the condition that caused the adaptation is no longer valid. This criteria is specially important when dealing with cumulative adaptations (*i.e.*, adaptations that can be applied over previous adaptations), since removing the changes of the first adaptation will alter the outcome of the later ones. This is explained in more detail in Section 4.3.

3.3.3 Discussion of horizontal adaptation

In this Section we have presented some of the work that has been done to obtain a horizontal adaptation of business processes. There are many different approaches to achieve this goal, however, thanks to its known capabilities to provide dynamic reconfiguration, the aspect-oriented approach is one of the preferred mechanisms, as can see in TAB. 3.2.

It is worth noting that in this case, not all of the presented solutions offer context integration, and as a consequence do not provide any automatic adaptation. Some of these approaches are mainly seen as a tool that allows the business process to be adapted, and leave the context monitoring and triggering of such adaptations to an external entity. Such was the case with the AO4BPEL solution presented in [Charfi and Mezini, 2007], which allowed a complete manipulation of the business process structure, but did not offer any context integration nor automatic adaptations. The authors noticed that there was a need for this integration and came up with their new approach in [Charfi et al., 2009], where they offer monitoring aspects.

Another thing to note is that, since the integration with context information is either limited or non-existent in most of these approaches, there is not enough information to do proactive adaptations, hence adapting only to correct a problem

Reference	Change service provider	Change activity behavior	Add activity	Delete activity	Context monitoring		Context scope	Automatic adaptation	Proactive adaptation	Adaptation undoing	Adaptation mechanism
					Service monitoring	Monitoring aspects					
Charfi et al., 2009	Yes	Yes	Yes	Yes	Policy rules	Constraints	Service context	Yes	No	No	Aspects
Geebelen et al., 2010	Yes	Yes	Yes	Yes	–	–	Business process	Yes	No	Yes/No	Aspects
Koning et al., 2009	Yes	Yes	Yes/No	No	–	–	–	No	No	No	JMX
Leitner et al., 2010	Yes	Yes	Yes/No	Yes/No	CEP-based	Events	Service context	Yes	Yes	No	Aspects
Marconi et al., 2009	No	No	Yes	No	Constraints	Events	Business process	Yes	No	No	Models
Sánchez and Villalobos, 2008	No	No	Yes	No	–	–	–	No	No	No	Aspects
Xiao et al., 2011	Yes/No	No	Yes	No	Policy rules	Constraints	Business process	Yes	No	Yes/No	Process Schemas
Zhai et al., 2008	Yes	Yes	Yes	Yes	Service monitoring	–	Service context	No	No	No	Reflection

Table 3.2: Summary of horizontal approaches

once it has arrived. Also, we can see that adaptation undoing is not really considered in most of the approaches, which could turn out to be a problem when dealing with multiple adaptations.

The works of [Geebelen et al., 2010] and [Xiao et al., 2011], both claim to support this task, however, in both cases it is a special kind of unadaptation. In [Geebelen et al., 2010], they do a rollback of the last adaptation, recovering its previous state when a policy rule is not satisfied, which limits the unadaptation capability to only one state if the adaptation did not turn out to be optimal. In the case of [Xiao et al., 2011], their adaptation is done by adding "groups of activities" called `fragments`, which they insert in specific points of the processes previously defined. For undoing the adaptations, they just remove the fragment from the `variation point`, which is like removing an activity (or several). Neither of them consider the undoing of cumulative adaptations, which would involve a more complicated logic.

Advantages of horizontal adaptation. By using horizontal adaptation of business processes, we have the advantage of being able to modify the process structure completely, adding new behavior or changing the existing one to respond better to the new needs, without the need to redeploy the process. Another advantage of this kind of adaptations is that it allows the separation of concerns, by deploying only the main business process and then adapting the process to cover different concerns. This can also help in the maintainability of the main processes, as they become less complicated. Finally, some horizontal adaptation mechanisms can also be used to make vertical adaptations, since the capability of changing the activity behavior (or even changing one activity for another one) can also be used to modify the service providers.

Disadvantages of horizontal adaptation. One of the disadvantages of these approaches is that, since BPEL engines are created to run static processes, and their goal is to modify the structure of the process, they need a customized engine that can support dynamic reconfiguration. This hinders their usability when trying to work with standard solutions, since the new engine should be able to deal with an extended syntax that considers the adaptation points (*e.g.,* a *join point* when using aspects). Another disadvantage is that adapting the structure of the process may consume more time than just doing vertical adaptation, however this is an arguable point, since the time it takes to adapt the process can be negligible depending on the kind of process we want to adapt. We have to consider that there are processes

that are completely automated and take some seconds to execute, while other can take several hours or even days. The more time a process takes to execute, the more negligible the adaptation time becomes.

3.4 Undoing adaptation

As we have presented before, dynamic adaptation has been widely considered, however, most of these approaches for adaptation are only one way, and they never consider undoing their changes. While creating dynamically adaptable business processes can be a challenging task, undoing these adaptations is a natural functionality that has not been studied in depth. Straight forward approaches for undoing an adaptation can easily end up with corrupted processes, bringing uncertainty to the whole business logic. Moreover, some adaptations may be related to previous ones, and determining how the undoing of an adaptation will impact the rest is not an easy task.

In this section we will present some of the few works done that consider a roll-back for their adaptations, nonetheless we will no longer be focusing only on business processes, since this is a general drawback of dynamic adaptation and not only of business process adaptation. At the end of this Section we will discuss about the need for undoing adaptations and the existing limitations.

3.4.1 Approaches for undoing adaptations

In [da Silva et al., 2010], the authors propose to automate the handling of model inconsistencies through the discovery of repair plans, implemented as action sequences. They demonstrate that action–based approaches support an *efficient* implementation of model manipulation. The Fractal component model includes a language called FScript [Léger et al., 2010], which uses actions to support automated rollback.

An aspect–oriented approach called WComp, is presented in [Tigli et al., 2009]. WComp is a lightweight component–based middleware to design composite Web services. The authors propose an aspect-oriented approach called *Aspect of Assembly* (AA) to create a *composition for adaptation*. When a change in the context is detected, they create a simulation by applying all the AAs (implementing the remaining adaptation rules) to the initial state and compare it to the actual state. Then

they apply the differences by using pure elementary modifications (add, remove, link, unlink).

The authors in [Klein et al., 2009] propose an action–based approach to support the unweaving of model aspects. The underlying principles are close to the ones used in this proposal, *i.e.*, the execution of inverted action sequences and the replay of remaining adaptations (in this case, aspect application). However, this approach cannot be applied dynamically, as aspect model unweaving is a human–driven process and needs to be triggered manually.

In [Bernal et al., 2010], the authors present their approach for creating dynamic business processes using ECA (Event-Condition-Action) rules. They decompose the original business process structure in a set of rules. This rules are then used to create a *Control Flow Checking* table, where the flow of the process is defined. To adapt the process they create a new modified *Control Flow Checking* table, which they compare to the original. The differences between both tables are then used to create new rules that will allow the new modifications to be considered during the business process execution. To undo the adaptation, the new rules could just be removed, or restored to their previous state.

3.4.2 Discussion of adaptation undoing

We have showed in this Section that undoing adaptations is something that is needed, but usually considered as something trivial and not really studied at depth. The presented works describe potential solutions to undoing adaptations, but they are focused on being used under very specific circumstances. For instance, the work of [da Silva et al., 2010] is only for recovering in case of a reconfiguration failure. If a reconfiguration is successful, the unadaptation is never used. In the case of [Tigli et al., 2009], they focus only on one kind of event (*i.e.*, service apparition or vanishing), which limits its usability. Their approach could certainly benefit from the use of a CEP-oriented solution to consider a wider variety of information.

Another limitation is some cases is the lack of dynamicity, as is the case of [Klein et al., 2009], where the unadaptation has to be manually triggered. The work of [Bernal et al., 2010] seems to correctly accomplish the goal of doing and undoing business process adaptations, however, the introduction and removal of the adaptation rules require external interaction, (*i.e.*, somehow the rules need to be created and fed into the system).

We believe that since undoing adaptations is a concern for all dynamic adaptation approaches, there should be a generic approach that helps to solve it, and which logic is not limited to a specific type of adaptation. Moreover, a correct adaptation undoing should also consider all the subsequent adaptations that happened after the first one, so that cumulative adaptations are not discriminated.

3.5 Challenges

After presenting some of the works in the area of business process and adaptation, we can justify the challenges that we had previously presented in Chapter 1. The goal of our work is to provide the business processes with enough flexibility to allow it to automatically and dynamically respond to the context changes in their execution environment. To achieve such a goal, we have defined three main challenges: ***i)* dynamic business process adaptation,** ***ii)* context integration,** and ***iii)* proper undoing of adaptations.**

3.5.1 Dynamic business process adaptation

As we could see with the presented works in this Chapter, dynamic adaptation has been widely recognized as a drawback of the existing implementations of business process management, either in the way of vertical or horizontal adaptations. For this adaptation to be really worthwhile, it has to really improve the outcome of our business processes, and should allow them to overcome any difficulties encountered during the execution.

Moreover, the adaptation should not only be done in a corrective manner, but also as a preventive alternative to avoid experiencing a possible problem, since it is usually less expensive to avoid a problem than it is to correct it. This means that not only should the adaptation be automatic, but also proactive. However, as we have seen in TAB. 3.1 and TAB. 3.2, most of the present works do not provide a proactive adaptation, which leaves a gap between the needs and the solutions.

3.5.2 Context integration

This second challenge is related to the first one. As we have stated, one of the requirements for a worthwhile dynamic adaptation is for it to be automatic and

proactive. To achieve this, we need to receive some kind of feedback about the state of the process. Furthermore, to actually be able to be proactive, we need to have access also to information outside the business process, that might influence its performance or its result.

This is the reason why we require the business processes to be integrated with the context information. The more sources of information concerning the execution of the business process, the better we will be able to prevent an undesired situation to happen, and also the better we will be able to respond if a bad scenario actually arrives.

In fact, there is already a tendency to integrate context information to business process management. For instance, in [Janiesch et al., 2011], the authors propose an architecture for an Event-Driven Business Activity Monitoring, integrating Business Process Management (BPM) and context information via Complex Event Processing (CEP), in a similar way to the Event-Driven Business Process Management work presented by [von Ammon et al., 2009]. In their work they accent the need of adding context awareness to the BPM and how this is still lacking in recent implementations. They base their architecture in the three main roles of event processing: event producer, event processor and event consumer. They show how a BPM can send information to the event processor (CEP engine), and at the same time profit from the resulting information of the processor after integrating external sources, as it can consider that information during the process execution.

However, even though considering the context information to be able to make the correct decisions during the execution, the response of the process to any changes in that information is limited to the predefined scenarios considered when creating the process, and any unexpected situation will have to be dealt in the traditional way of stopping the process, modifying it and finally redeploying it with a new behavior. This is something that can be complemented with a dynamic business process adaptation approach, in which case the changes in the context information can be considered at run-time without redeploying the process.

3.5.3 Correctly undoing adaptations

Just as the second challenge, this third challenge is also related to the first one. As we presented in Section 3.4, undoing an adaptation has not been widely studied, as it is naively considered as a trivial task of just removing the changes done to

the system (or process in this case). However, as we will show in Chapter 4, this is far from true, and we need to pay special attention when undoing cumulative adaptations.

Most of the works presented in Sections 3.2 and 3.3, do not consider any mechanism to undo their changes, and those who do are actually implementing a workaround for this issue. To properly undo an adaptation, we need to consider the state of the process before that adaptation occurred, but also all the events and adaptations that came after it, and not only restore the process to its previous state (before the adaptation occurred).

3.5.4 Intended solution

Taking into consideration the criteria used above to evaluate the current State of the Art, our goal is to create a solution that tackles the aforementioned challenges. For this, it would have to be able to dynamically modify the business process by **adding** new activities to the process, and **removing** or **updating** the existing ones.

Moreover, the solution should be able to **monitor** the current state of the context, with a scope that is not limited only to the business process nor to the business logic, but that can consider the **context in general**. The monitoring of the context should allow it to detect the special situations when an adaptation is needed, so that the process can be **automatically adapted**. Also, the general scope of the context monitoring should allow it to generate not only automatic, but also **proactive adaptations**, as it may infer incoming situations using the context information.

Finally, this solution should also consider a mechanism to correctly and **automatically undo** the created adaptations, when the condition that triggered the adaptation is no longer valid. The summary of this characteristics is presented in TAB. 3.3.

	Add / Update / Delete	Context monitoring	Context scope	Automatic adaptation	Proactive adaptation	Adaptation undoing
Our goal	Yes/Yes/Yes	YES	General	Yes	Yes	YES

Table 3.3: Intended solution

3.6 Summary

In this chapter we have presented several approaches for adapting business processes. We grouped all of the approaches into two groups: vertical adaptations and horizontal adaptations. These two groups, both aim at adapting the business process, but with different goals in mind. While the vertical adaptation approaches aim at adapting the service providers, the horizontal approaches try to modify the behavior of the whole process. This makes both of these approaches to be complementary, as they do not really compete to achieve the same goal, other than improving the business process execution.

We have also shown that undoing the adaptations is still being left behind, and even though some works have been done to achieve this, none has actually been implemented for the business process domain. Finally, we revisited the challenges previously introduced in Chapter 1, and justified them with the analysis of the different approaches presented here.

With this we conclude the part of this dissertation about the State of the Art. In the following chapters we will present CEVICHE, our contribution towards creating context-aware dynamically-adaptable business processes using complex event processing.

Part II

Contribution

Chapter 4

Event-based Dynamically-adaptable Business Processes

"Learning is never done without errors and defeat."
- Vladimir Lenin

Contents

The contribution of this work is divided in two parts:

1. In the first part, Chapter 4, we present our solution for dynamically adapting business processes using an event-driven approach to provide context information. In this chapter, we also show how undoing these adaptations is not a trivial task and present our proposal for correctly undoing an event-based adaptation, in a clean and automatic way.

2. In the second part of our contribution, in Chapter 5, we present our implementation for doing and undoing dynamic adaptations presented in Chapter 4, called the CEVICHE Framework. We use a component-based approach to provide dynamicity to business processes and Complex Event Processing as a way to deal with context information.

4.1 Introduction

Nowadays, there is a huge amount of data surrounding our business processes, which results in lots and lots of variable conditions that may affect their outcome. Given the huge increase in the use of distributed computing, plus the number of event sources that are available nowadays, the working environment of the organizations is becoming more and more dynamic. The context in which our business processes are executed is an important factor, and we need to be able to monitor it so that it can be considered during the execution. By monitoring the context in which business processes are being executed, it is possible to efficiently respond to any changes in the environment and continue the process in an optimal way.

Changes in the context can be seen as something that happens at a specific moment in time and have a different meaning depending on several conditions, *e.g.*, timing, origin, sequence [Adi and Etzion, 2004]. The conditions under which we consider these changes can be interpreted using one or more of the following domains: temporal context, spatial context, segmentation-oriented context and state-oriented context, as previously explained in Section 2.3.

To consider the context information in a business process, we would need to define some specific moments during its execution at which this information will be evaluated and considered to make a decision about how to continue. Unfortunately, the static nature of business processes makes it difficult to consider all the

context information at every step of its execution. Moreover, we need to anticipate all the possible situations that might occur at those specific decision moments, and all the possible answers to them, since once the process is deployed, it cannot be modified.

Very often, unpredictable situations happen and errors occur, creating a need for dynamic changes, since it is not always feasible to stop the execution of a running process and then redeploy it, because that would cause the loss of all the current information, which is specially harmful when executing long-running processes (*i.e.,* processes that can take several hours or days to execute). Additionally, when referring to business process execution, we rely on external sources to provide the Web Services of the process, which are not always reliable as they are out of the control of the process owner, and so, for all these motives, adaptability has become a very important non-functional requirement.

In this chapter we present a formalized approach to solve these issues by creating event-based dynamic adaptations for business processes. We make use of complex events to determine specific situations that require an adaptation and then use simple functions (add and delete) for modifying the business process. However, there is another issue which is also important and that is usually (but naively) considered as a trivial task. This is, undoing the adaptations when the conditions that led to them are no longer valid. Usually, undoing an adaptation is thought as just doing the opposite operations required for doing it, but as we will show in the following sections, it is a much more complex task than that, and if it is done without precaution, it can lead to instable and/or corrupted processes.

Structure of the Chapter

The rest of this Chapter is organized as follows: First, in Section 4.2 we present how we deal with business process adaptation using an event-driven approach. Next, in Section 4.3 we show why undoing adaptations is not a trivial task and present our approach for properly doing it. Finally, Section 4.4 summarizes the ideas presented in this Chapter.

4.2 Adaptation in Business Processes

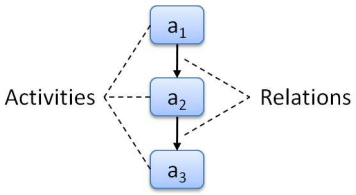
According to [Courbis and Finkelstein, 2005], adaptations can be evaluated using three different classifications: i) static or dynamic, ii) manual or automatic, and iii) proactive or reactive.

- The first classification refers to how the changes to the applications are executed. Static adaptations are carried out through modifications in the source code, before the system is running, while dynamic adaptations modify the software characteristics at run-time.
- The second classification refers to the procedures used to manage the adaptations. Manual adaptations require direct human intervention in the system to indicate when the modifications should be considered, while in the automatic ones the system itself detects the conditions needed for the adaptation and executes the modifications.
- Finally, the third classification refers to the moment in which the adaptation takes place. In this case, proactive adaptations occur before a specific event arrives, for example, we can prevent a specific situation (*e.g.,* a downtime in the service) by monitoring the surrounding events that may indicate that it is about to happen (*e.g.,* excessive CPU use, long response times, etc.) On the other hand, reactive adaptations happen after the event has been detected, as a response to or consequence of it, for example, we could use the adaptations in a reactive manner by changing the behavior of the application once we have encountered a specific situation (*e.g.,* service unavailable).

When referring to business processes, static adaptations would mean to modify the process and redeploy it to the execution engine, which will lead to loosing all the current process instances, as well as all of their information, which is already a big loss without considering the downtime of the service while the process is being redeployed. In Chapter 3 we presented some of the existing solutions to prevent this problem by using dynamic adaptation, however we also saw that there are still certain limitations with these approaches for the modern environments. In this section, we will present how context information can be interpreted using events, and how these events can be used to dynamically adapt business processes as a response to changes in the execution environment.

4.2.1 Business Processes & Actions

In order to present how adaptations can be integrated into business processes, let us begin by formally defining a business process. We define $\mathcal{P}$ as the set of business processes, where p is a business process that belongs to that set. We may define $p \in \mathcal{P}$ as a set of activities $acts$, which implement elementary tasks, and a set of causal relations $rels$ to schedule the activity set according to a partial order. $\mathcal{R}$ is the set of all the relations (or links) in $\mathcal{P}$. To simplify the understanding of the process, we assimilate an activity to its $name$, without further knowledge of its internal contents. A (binary) causal relation is defined as an ordered pair of activities (*i.e.*, $left$ and $right$). We denote as $left \prec right \in \mathcal{R}$ the fact that a relation exists between $left$ and $right$. We depict in FIG. 4.1 a business process and its associated formal representation.



$$p = (\underbrace{\{a_1, a_2, a_3\}}_{acts}, \underbrace{\{a_1 \prec a_2, a_2 \prec a_3\}}_{rels})$$

Figure 4.1: A simple business process, $p \in \mathcal{P}$.

To manipulate business processes, we use an action–based approach, since these approaches are known to efficiently support the manipulation of models [Blanc et al., 2008]. These actions are provided by the user in an ABPL definition. An elementary action is defined as the addition or deletion of an element in a given business process. In itself, an action α is simply a ground term that reifies the associated intention (*e.g.*, adding an activity, deleting a relation). In TAB. 4.1 we can see a list of the existing actions available to modify a given business process. In our approach, whenever an activity with existing relations is removed, those relations are reestablished between its predecessors and its successors. The execution of an action α on a process p is handled by a call to the $exec$ function: $exec(\alpha, p) = p'$, where p' is a process effectively modified.

Actions can be sequenced to implement complex modifications. Let $A = [\alpha_1, \ldots, \alpha_n]$ be a sequence of actions. We assimilate a sequence to a totally ordered

Intention	Notation
Add an activity a	$add_a(a)$
Add a relation $a \prec a'$	$add_r(a, a')$
Del an activity a	$del_a(a)$
Del a relation $a \prec a'$	$del_r(a, a')$

Table 4.1: Actions available to manipulate business processes

set (*i.e.,* a list), and use the notation and functions associated to lists usually encountered in the logic programming literature [O'Keefe, 1990]. A list L is defined as a head h followed by a tail list T, and is denoted as $L = [h|T]$. The empty list is $\emptyset$. The execution of A on a given process p is formally defined as follows:

$$exec^+(L, p) = \begin{cases} L = \emptyset & \Rightarrow \quad p \\ L = [\alpha|A] & \Rightarrow \quad exec^+(A, exec(\alpha, p)) \end{cases}$$

Using this representation, the process p depicted in FIG. 4.1 can be built as the result of the execution of its associated action sequence A_p on the empty process.

$$\begin{aligned} A_p &= [add_a(a_1), add_a(a_2), add_r(a_1, a_2), add_a(a_3), add_r(a_2, a_3)] \\ p &= exec^+(A_p, (\emptyset, \emptyset)) \end{aligned}$$

4.2.2 Events & Context-awareness

As mentioned earlier in Chapter 2, context-awareness refers to an application's ability to react to changes in the environment and use context information during its execution. To trigger the actions that will allow us to manipulate the business process, we need our processes to be context-aware, and so we have to provide them with a way to receive information from its environment. For this, we will use an event–driven approach, where the information coming from the environment is represented as events. The most general definition we can find about an event is:

> *"An event is anything that happens."* [Chandy and Schulte, 2010]

We can consider an event as, for example, an update of a database, a state of change in a process, a reported problem or any business situation raised by an application. An event is a significant atomic occurrence in the reality or virtual reality. It is significant in the sense that it may affect some action, atomic because it happens completely or not at all, and an occurrence as it is contemplated as happening, as it could be a fact becoming true or a state transition. A more extensive definition of an event would be:

> *"An event is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word event is also used to mean a programming entity that represents such an occurrence in a computing system."* [Etzion and Niblett, 2010].

In this definition we can find two meanings for an event; one that refers to an actual occurrence (the something that has happened) in the real world or in some other system, and a second that refers to event processing, where an event is meant as a programming entity representing this occurrence. A single event occurrence can be represented by many event entities, and a given event entity might capture only some of the facets of a particular event occurrence.

We can define $\mathcal{E}$ as the set of events occurring in the environment of a process p. We may define $\epsilon \in \mathcal{E}$ as a tuple <*name*, Q>, where *name* is the name of the event, and Q is a set of attributes contained in the event. At the same time, we define $q \in Q =< attribute, value >$, where $attribute$ is the name of the attribute, and $value$ is the assigned value of that attribute. These attributes can be, *e.g.*, the event's source, its time of creation, etc.

To obtain the information needed from those events, we use *event processing*. Event processing can be defined as the tuple <$\mathcal{E}$, P, C, Ch>, as is illustrated in FIG. 4.2, where:

- $\mathcal{E}$ is a set of events in the environment.
- P (*event producers*) is a set of entities that introduce events into the system.
- C (*event consumers*) is a set of entities that receive events from the system.
- Ch (*event channels*) is a set of processing elements that receive events from one or more elements $p \in P$, make routing decisions and send the unchanged events to one or more elements $c \in C$.

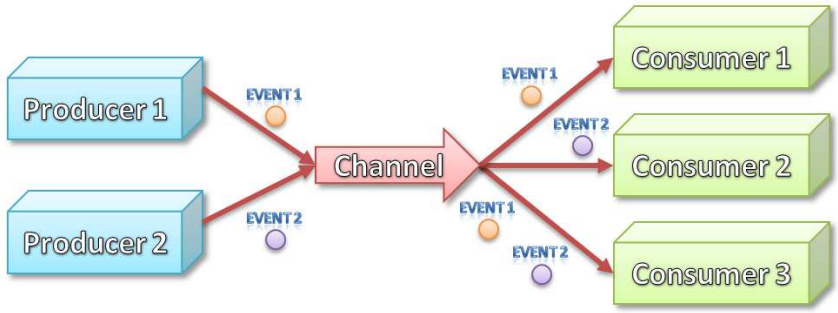


Figure 4.2: Example of Event Processing components

All events in the environment can be classified as either of two types: *raw events* ($\epsilon \in \mathcal{R}$), which are the simple events generated by an event producer, or *derived events* ($\epsilon \in \mathcal{D}$), which are generated as a result of processing one or more raw events. This classification can be expressed as:

$$(\forall \epsilon | \epsilon \in \mathcal{E} \wedge (\epsilon \in \mathcal{R} \vee \epsilon \in \mathcal{D}) : (\mathcal{E} \equiv \mathcal{R} \cup \mathcal{D}))$$

Any entity x that creates derived events has to be at the same time an event producer and an event consumer, which is expressed as:

$$(\exists x | : x \in P \wedge x \in C)$$

The way in which events are processed often requires taking into consideration the context in which the events occurred. This context can involve segmentation, location, time, sequence, and/or the state of an external entity [Chandy et al., 2011]. To relate these events, we need to explore temporal, causal, and semantic relationships among them, in order to make sense out of them in a timely fashion. This reveals opportunities and threats as soon as they emerge or can serve to diagnose and execute decisions in a time constrained fashion [Luckham, 2002].

The relation among the events can be specified using *Complex Event Processing* (CEP). In our approach, *complex events* (CE) are defined as *(i)* a boolean formula applied to an (elementary) event to process it or *(ii)* a combination of other complex events. We represent in TAB. 4.2 the expressiveness associated to usual complex event definitions. Complex events can be conjucted ($\wedge$) or disjuncted ($\vee$) using elementary boolean logic. Also, a sequence operator can be used to introduce causality between two events ($\epsilon_1; \epsilon_2$ means that ϵ_1 is eventually followed by ϵ_2, even if

not immediately). Finally, a time window operator supports the wait for a given complex event during a given amount of time (*e.g.*, $\epsilon' = within(\epsilon, 200ms)$ will be recognized if the complex event ϵ is received by the engine within $200ms$).

Intention	Notation	Example
Event processing	$(attribute \sim value)$	$(cpu{>}80\%)$
CE conjunction	ϵ_1 ***and*** ϵ_2	$slow$ ***and*** $(error{=}404)$
CE disjunction	ϵ_1 ***or*** ϵ_2	$(error{=}404)$ ***or*** $(error{=}503)$
CE sequence	$\boldsymbol{seq}(\epsilon_1; \epsilon_2)$	$\boldsymbol{seq}(fail;slow)$
Time window	$\boldsymbol{within}(\epsilon, \Delta_t)$	$\boldsymbol{within}(\neg response,10s)$

Table 4.2: Complex Event Definitions

4.2.3 Event–driven adaptation

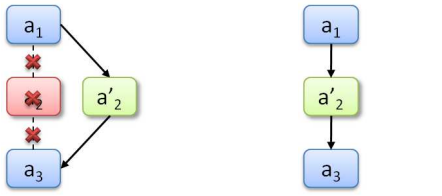
To add adaptability into existing business processes, we first need to define some rules that will indicate how such an adaptation will be executed. We call these *adaptation rules*. An adaptation rule $r \in \mathcal{A}_\mathcal{R}$ is defined as a tuple (ϵ, φ), where ϵ is the complex event used to trigger the adaptation, and φ is a function used to compute the action sequence to be executed to perform the adaptation. According to state–of–the–art engines, we assume that an adaptation is only triggered once. This action sequence is executed on the business process, to modify its structure and then implement the adaptation:

$$(\epsilon, \varphi) \in \mathcal{A}, p \in \mathcal{P},\ \epsilon \Rightarrow exec^+(\varphi(p), p)$$

We illustrate such an adaptation in FIG. 4.3. The goal of this adaptation is to replace an activity by another one when the complex event ϵ is processed. The application of φ on the process p produces a sequence of actions A, which aims to replace the activity a_2 by a new activity a_2'. To implement this adaptation, the engine executes A on p, and computes as output p', the adapted process.

When defining these adaptations, there are certain considerations that need to be taken into account. First, we can consider that a situation $\sigma \in \mathcal{S}$ defines only one specific circumstance linked to one specific response (as an adaptation A) of the business process.

$$(\forall p| : (\exists \sigma, A| : \sigma \Rightarrow exec^+(A, p)))$$



(a) $\varphi(p) = A$ (b) $p' = exec^{+}(A, p)$

$$A = [add_a(a'_2), add_r(a_1, a'_2), add_r(a'_2, a_3), del_r(a_1, a_2), del_r(a_2, a_3), del_a(a_2)]$$

Figure 4.3: Applying an adaptation (ϵ, φ) to p

Then, we consider $\mathcal{D} \subseteq \mathcal{E}$, where $\mathcal{D}$ is a subset of all events in $\mathcal{E}$ which would trigger a specific situation σ. This can be defined as:

$$\mathcal{D} = \mathcal{E} \cap \mathcal{A}_{\mathcal{R}}$$

Finally, we consider that a specific situation $\sigma \in \mathcal{S}$ can be triggered by many different events ($\delta, \delta' \in \mathcal{D}$), and that an event $\delta \in \mathcal{D}$ may trigger many different situations ($\sigma, \sigma' \in \mathcal{S}$). This is defined in the following way[12]:

$$(\forall \sigma \mid : (\exists \delta, \delta' \mid : ((\delta \Rightarrow \sigma) \wedge (\delta' \Rightarrow \sigma))))$$
$$(\forall \delta \mid : (\exists \sigma, \sigma' \mid : ((\delta \Rightarrow \sigma) \wedge (\delta \Rightarrow \sigma'))))$$

4.2.4 Adaptation Example

To show how the adaptation of a business process is achieved, we consider here a simple process, part of an online catalog software. It contains five activities, which respectively: *(i)* logs the user in, *(ii)* asks for user's request, *(iii)* performs the search in the internal database, *(iv)* displays the results to the user and finally *(v)* logs the user out. This process is depicted in FIG. 4.4.

[12]To note that in both situations, (δ, δ') as well as (σ, σ'), the instances are not required to be different, and may therefore refer to the same instance respectively.

Figure 4.4: Illustrative business process (initial)

We want to adapt this process according to the context, using an event-driven approach. Process adaptations are driven by the reception of explicit complex events (triggered by associated conditions). For example, if the search service becomes unavailable, a *fail* event will be triggered, and an adaptation will be executed to fix the problem. Precisely, it will connect the process to a remote backup service, to ensure continuity for customers. We summarize in TAB. 4.3 the different adaptation rules associated to our running example.

Event	Condition	Action
fail	$search\_status \neq ok$	Use a backup server
slow	$bw < 100kbps$	—
cache	**fail** *followed by* **slow**	Introduce a cache
perf	$cpu > 80\%$	Monitor the process

Table 4.3: Event–driven adaptation decisions

Accordingly, if the *fail* event is received, the business process will be adapted to solve this problem, and we will obtain after the adaptation the process depicted in FIG. 4.5. In this figure (and the upcoming ones), we represent deleted elements with *dashed* lines.

Figure 4.5: Consulting a backup when the search service is unavailable

If we then receive a performance alert by the event *perf* (identifying a CPU abnormal usage), we want to start monitoring the CPU consumption for all the activities in the process. To achieve this, we will add a monitoring activity after each existing activity. The resulting process is depicted in FIG. 4.6.

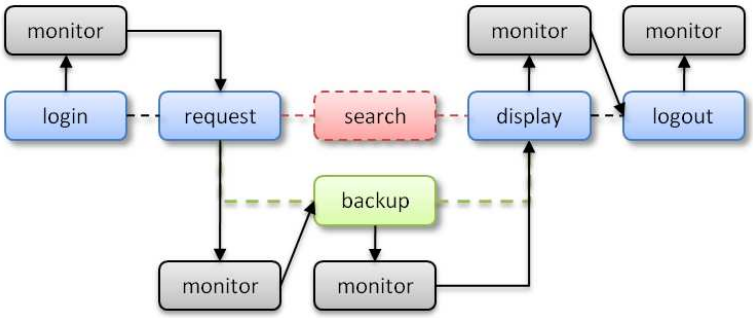


Figure 4.6: Monitoring the process to identify abnormal CPU consumption

Since the backup server is a remote entity, we depend on the quality of the network connection to search the catalog. Considering a bandwidth drop below 100kbps (identified with a *slow* event), the *cache* event will also be recognized (as it is defined as an event *fail* followed by an event *slow*) and we will need to adapt the process by adding a cache mechanism to help diminish the response time of the requests. If the event *fail* had not occurred, the arrival of the *slow* event would have no relevance on the process (because there would not be a remote server) and hence would not cause any adaptation to be triggered. The adapted process is depicted in FIG. 4.7.

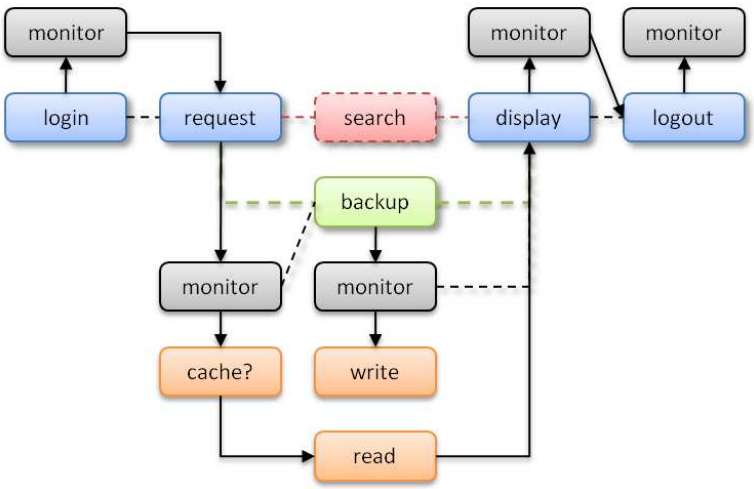


Figure 4.7: Introducing a cache to deal with lower bandwidth

4.3 Undoing Process Adaptations

While creating dynamically adaptable business processes can be a challenging task, undoing these adaptations is a natural functionality that has not been studied in depth. Straight forward approaches for unadaptation can easily end up with corrupted processes, bringing uncertainty to the whole business logic. The goal of this Section is to present an effective solution to event–driven business process unadaptation, by considering not only the event that caused the adaptation, but also the correlated adaptations that came afterwards, leaving all the unrelated adaptations untouched, in order to obtain a business process *"as it would be if this adaptation had never happened"* [Mosser et al., 2011] (similarly to transactional systems [Bernstein et al., 1987] where the *rollback* operation is used to restore a system). Using this generic and automated approach, users are relieved from handling the unadaptation logic.

4.3.1 Need for Adaptation Undo

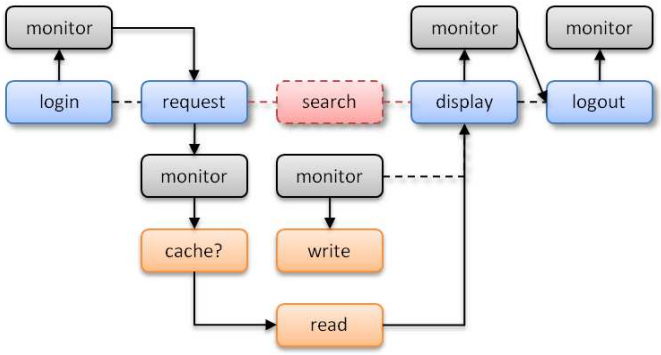
When an adaptation condition is no longer true, we would like to get our process as it would be if this adaptation had never happened. Retaking our previous example, let's say we receive an event $\neg$*fail*, which means that we recovered our internal search server. In this case, we no longer need the external backup nor the associated cache mechanism and we can remove them.

We represent in TAB. 4.4 the way an oposite event $\neg\epsilon$ is computed with respect to an event ϵ.

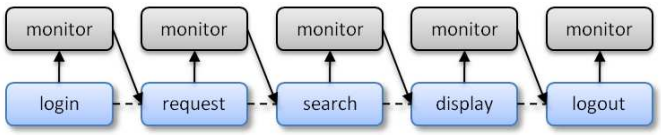
Complex Event (ϵ)	Opposite Event ($\neg\epsilon$)
$(attribute \sim value)$	$(attribute \not\sim value)$
$\epsilon_1 \wedge \epsilon_2$	$\neg\epsilon_1 \vee \neg\epsilon_2$
$\epsilon_1 \vee \epsilon_2$	$\neg\epsilon_1 \wedge \neg\epsilon_2$
$\epsilon_1 ; \epsilon_2$	$\neg\epsilon_1 \vee \neg\epsilon_2$
$within(n, \epsilon_1)$	$\neg\epsilon_1$

Table 4.4: Complex Events (ϵ) & Opposites ($\neg\epsilon$)

Naively, undoing an adaptation does not seem so complicated. It can be seen as removing all the changes made to the business process that were caused by the *fail* event. In order to achieve this, the intuitive undoing action would be to use the exact "opposite" of the used adaptation. In our case, it would remove the backup server and re–introduce the internal search one. The associated process is depicted in FIG. 4.8(a). Unfortunately this process does not make sense in terms of business logic, as it holds the two following issues: *(i)* the *search* activity is not monitored and *(ii)* the cache mechanisms are irrelevant since the vanishing of the backup server. Syntactically speaking, the removal of the backup activity also creates a hole between the cache validity test and the cache writing activity, leading to a corrupted process.



(a) Inconsistent process obtained after naive unadaptation



(b) Expected result

Figure 4.8: Undoing adaptation (¬*fail*): a *not–so–easy* task

4.3.2 Mechanisms for Proper Unadaptation

As seen in the previous example, a straight forward undoing of the adaptation can result in a corrupted process. This risk is even higher as the business processes get bigger and more complex. To obtain a correct undoing of an adaptation, we could

add an adaptation rule that changes the process to its original state. However, this approach will only work if we consider all the possible states of the process, given all the possible adaptations that could happen, providing the correct process for each and every one of them. This, far from being user friendly, is virtually impossible to accomplish.

To tackle these issues, we propose to automate the support of business process unadaptation. The key idea is to formalize the adaptation, and to rely on this formal model to define and then operationalize the unadaptation. We consider here an event–driven adaptation engine based on state–of–the–art mechanisms [Sharon and Etzion, 2008], represented in FIG. 4.9. At a coarse–grained level, the engine receives a continuous flow of events from deployed sensors. According to the received events, the CEP engine will trigger the associated adaptations, stored in an adaptation repository. The obtained (*adapted*) process is then sequentially used as input for the upcoming adaptations.

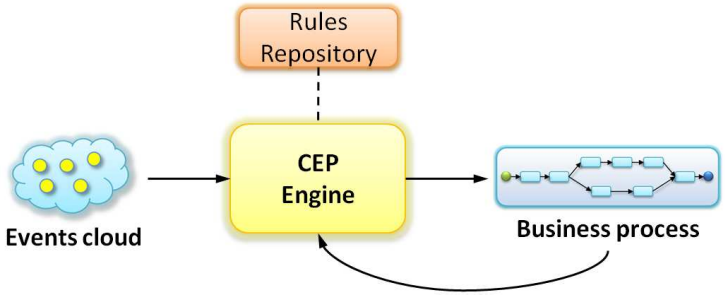


Figure 4.9: Overview of the adaptation process

To properly support unadaptation of business processes, we need to keep track of the adaptation history. This concept is expressed as a list of tuples (ϵ, A_ϵ), where ϵ is the processed complex event and $A_\epsilon = [\alpha_1, \ldots, \alpha_n]$ the sequence of actions computed according to this event (as previously presented in Section 4.2.1). The list is maintained in reversed order, *i.e.*, the head of the history corresponds to the last adaptation. Considering the final adapted process (FIG. 4.7) of the example, the history is defined as follows:

$$H_{ex} = [(perf, A_{perf}), (slow, A_{slow}), (fail, A_{fail})]$$

The ideal case would be to provide the user with an automated unadaptation of the process, whenever the adaptation conditions are no longer met. Using this approach, we could automatically produce a system as it would be if this adaptation had never happened. Going back to the example of Section 4.2.4, the result of this approach can be seen in FIG. 4.8(b): the search activity is monitored, and the cache mechanism is not present (since its trigger event depends on the *fail* one). To support this automated unadaptation, our approach uses the following four mechanisms M_i, which are required to properly undo an adaptation triggered by an event ϵ:

- M_1: **Identify the undoing trigger**. Based on the description of events, the system must be able to recognize their opposite, and trigger the automated undoing mechanisms when relevant. In TAB. 4.4 (p. 71) we can see how the system determines if an event is the opposite of a previous one. The opposite events are only monitored for the active events in the history.

- M_2: **Restore the process**. The current process must be restored to what it was *before* the reception of ϵ. This means that we would have to go back in the history and undo all the changes made to the process up to the point when the event arrived.

- M_3: **Forget the correlated adaptations**. Adaptations triggered by any event which depends (immediately or transitively) on ϵ must be forgotten. In this case we need to analyze all the history after the restoration point and eliminate all events that are related to ϵ (*e.g.*, the **`cache`** event depends on the **`fail`** event).

- M_4: **Re–execute the unrelated adaptations**. Finally, all adaptations that are independent of ϵ must be re–executed, to yield a system equivalent to the one obtained after their on–the–fly execution. We go again through the history of events, without considering the events related to ϵ, and readapt the business process.

4.3.3 Automating Adaptation Undoing

We consider here the situation depicted in FIG. 4.10. This situation is a formal representation of the adaptation example textually described in Section 4.2.4: p is the initial scenario (FIG. 4.4), and p_{123} is the resulting adapted process (FIG. 4.7).

$$p \overset{\epsilon_1}{\longmapsto} p_1 \overset{\epsilon_2}{\longmapsto} p_{12} \overset{\epsilon_3}{\longmapsto} p_{123}$$

Defined complex events: $\{\epsilon_1, \epsilon_2, \epsilon_3\}$

Complex events combination: $\epsilon_3 = \epsilon_1; \epsilon_2$

Rule repository: $\{(\epsilon_1, \varphi_{\epsilon_1}), (\epsilon_2, \varphi_{\epsilon_2}), (\epsilon_3, \varphi_{\epsilon_3})\}$

Adaptation steps:

- $p_1 = exec^+(A_{\epsilon_1}, p),\ A_{\epsilon_1} = \varphi_{\epsilon_1}(p)$
- $p_{12} = exec^+(A_{\epsilon_2}, p_1),\ A_{\epsilon_2} = \varphi_{\epsilon_2}(p_1)$
- $p_{123} = exec^+(A_{\epsilon_3}, p_{12}),\ A_{\epsilon_3} = \varphi_{\epsilon_3}(p_{12})$

History: $[(\epsilon_3, A_{\epsilon_3}), (\epsilon_2, A_{\epsilon_2}), (\epsilon_1, A_{\epsilon_1})]$

Figure 4.10: Doing adaptation: p becomes p_{123}.

Doing adaptation. The adaptation rule repository holds three rules, defined with respect to three complex events: $\{\epsilon_1, \epsilon_2, \epsilon_3\}$. Complex events ϵ_1 and ϵ_2 come from elementary event processing, and ϵ_3 is defined as a sequence of events (the detection of ϵ_2 *after* the detection of ϵ_1). Based on the analysis of the incoming elementary events, adaptations are triggered using the information obtained from the complex event processing engine, to adapt a given process p. We consider here the following sequence of events: ϵ_1, ϵ_2, and consequently ϵ_3 (according to its definition). After these three adaptations, we obtain a process p_{123}. This process is handled through the previously defined mechanisms.

Undoing adaptation. We consider now the detection of a complex event opposed to ϵ_1 (denoted as $\neg\epsilon_1$). In this new context, adaptations that had been triggered based on ϵ_1 do not make sense anymore, and must be undone. Considering that our objective while undoing adaptation is to produce the system as it would be if the adaptation had never happened, we also need to undo all adaptations depending on ϵ_1 (*i.e.*, triggered by a complex event which combines ϵ_1 with others, here ϵ_3). According to this goal, and with respect to the mechanisms M_i previously presented, the system needs to *(i) recognize* the opposite event and then trigger

the undoing mechanisms (M_1), *(ii) rewind* the history of events to retrieve the process as it was before the reception of the incriminated event (M_2), *(iii) prune* from the history the adaptations that depend (immediately or transitively) on this event (M_3), and finally *(iv) replay* the remaining adaptations to obtain the expected process (M_4).

Considering the example depicted in FIG. 4.10, the undo mechanism associated to ϵ_1 is expected to automate the following steps:

- *recognize*: Assuming that ϵ_1 is an arithmetic comparison (*e.g.*, $bandwidth < 100kbps$), its opposite can be automatically computed (*i.e.*, $\neg\epsilon_1 = bandwidth \geq 100kbps$). An adaptation needs to be undone since the complex event processor engine detects the event and its opposite in sequence, *i.e.*, $\epsilon_1^{-1} = \epsilon_1; \neg\epsilon_1$. It is clear that before the reception of ϵ_1, the event $\neg\epsilon_1$ is not monitored and is hence being ignored by the system. The only opposite events that are monitored are the ones from the events present in the repository that are *active* (*i.e.,* that have been used in an adaptation in the current state of the business process).

- *rewind*: On the detection of ϵ_1^{-1}, the system will restore the process as it was before the detection of ϵ_1. In our case, this *rewind* restores the process p_{123} as p.

- *prune*: Considering the contents of the history, the engine knows that the process p was adapted according to the following sequence of events: $[\epsilon_1, \epsilon_2, \epsilon_3]$. The pruning step removes from this sequence the incriminated event ($[\epsilon_1]$), and all its (transitive) dependencies (here, ϵ_3, since it depends on $[\epsilon_1]$). In our case, the pruned sequence is $[\epsilon_2]$.

- *replay*: The adaptations triggered by the events contained in the pruned sequence need to be replayed in the process. In our case, it means to adapt p according to the rule associated to ϵ_2.

Once we go through all of the four steps of the adaptation undoing process, we obtain the newly unadapted business process p_2, as represented in FIG. 4.11.

$$p \underbrace{\xmapsto{\epsilon_1} p_1 \xmapsto{\epsilon_2} p_{12} \xmapsto{\epsilon_3}}_{do} p_{123} \underbrace{\xmapsto{\neg\epsilon_1} (p \xmapsto{\epsilon_2})}_{undo} p_2$$

$$p_2 = exec^{+}(\varphi_{\epsilon_2}(p), p)$$

Figure 4.11: Undoing adaptation: p_{123} becomes p_2

4.3.4 "Undo" Operationalization

In this section, we formally describe how the undaptation can be operationalized. We present the different operations used to support the $undo$ process using a functional style, being consequently language independent. We also apply each of the unadaptation mechanisms to the example, and show how using our approach, a correct undoing of the adaptation can be achieved.

4.3.4.1 M_1: Recognition of an Undo Trigger (ϵ^{-1})

We denote as ϵ^{-1} the complex event that triggers an undo. This event is defined as the sequence composed by the event ϵ and its associated opposite event $\neg\epsilon$. Using this definition, an undo will always be triggered when the engine recognizes an opposite event (*e.g.*, $\neg fail$) eventually preceded by an event (*e.g.*, *fail*).

$$fail^{-1} = fail; \neg fail$$

In TAB. 4.5 we show the events and conditions used in the example, and their corresponding opposites.

Event	Condition	Opposite Event	Opposite Condition
fail	$search\_status \neq ok$	¬**fail**	$search\_status = ok$
slow	$bw < 100kbps$	¬**slow**	$bw \geq 100kbps$
cache	**fail** *followed by* **slow**	¬**cache**	¬**fail** $\vee$ ¬**slow**
perf	$cpu > 80\%$	¬**perf**	$cpu \leq 80\%$

Table 4.5: Events and Conditions Opposites

4.3.4.2 M_2: Rewinding a Business Process

The objective of this function is to restore a process as it was before the reception of the initial event ϵ. It intensively relies on the history model, previously defined in Section 4.3.2, identifying the actions to be undone *and* the encountered events.

Undoing actions. For each kind of action α, we present in TAB. 4.6 its inverse α^{-1}. Executing α^{-1} after α annihilates the introduced modification: $exec(\alpha^{-1}, exec(\alpha, p)) = p$. Considering a sequence of actions A, its inverse (denoted as A^{-1}) is defined as the inverse of all actions contained by A, in reversed order. This approach is inspired by aspect unweaving techniques [Klein et al., 2009].

α	α^{-1}
$add_a(a)$	$del_a(a)$
$add_r(a, a')$	$del_r(a, a')$
$del_a(a)$	$add_a(a)$
$del_r(a, a')$	$add_r(a, a')$

$$\begin{aligned} A &= [\alpha_1, \ldots, \alpha_n] \\ A^{-1} &= [\alpha_n^{-1}, \ldots, \alpha_1^{-1}] \end{aligned}$$

Table 4.6: Actions (α) & Inverse (α^{-1})

In the example, we would have to undo all the modifications done to the business process before the arrival of the event *fail*. The sequence of actions A^{-1} would be [$del_a(cache)$, $del_a(monitor)$, $del_a(backup)$, $add_a(search)$] (the adding and removing of the links to those activities were omitted on purpose to simplify the explanation).

Function description. This operation is implemented in a $rewind$ function. Based on a given process p, the associated history H and the intended event ϵ, this function computes a process p' (representing the business process p as it was before the reception of ϵ) and a list of complex events $[\epsilon_i, \ldots, \epsilon_j]$ (representing all the events recognized between the reception of ϵ and $\neg\epsilon$). For clarity reasons, we decouple

the computation of p' (using a $restore$ function) from the identification of the encountered events (using an $extract$ function). The definition of these functions is presented in FIG. 4.12.

$$
\begin{aligned}
rewind : \mathcal{P} \times History \times \text{CE} &\rightarrow \mathcal{P} \times [\text{CE}] \\
(p, H, \epsilon) &\mapsto (p', [\epsilon_i, \ldots, \epsilon_j]) \\
restore : \mathcal{P} \times History \times \text{CE} &\rightarrow \mathcal{P} \\
(p, H, \epsilon) &\mapsto p' \\
extract : History \times \text{CE} &\rightarrow [\text{CE}] \\
(H, \epsilon) &\mapsto [\epsilon_i, \ldots, \epsilon_j]
\end{aligned}
$$

$$
restore(p, H, \epsilon) = \begin{cases} H = \emptyset \Rightarrow p \\ H = [(\epsilon, A_\epsilon)|H'] \\ \quad \Rightarrow exec^+(A_\epsilon^{-1}, p) \\ H = [(\epsilon', A_{\epsilon'})|H'], \epsilon), \epsilon' \neq \epsilon \\ \quad \Rightarrow restore(exec^+(A_{\epsilon'}^{-1}, p), H', \epsilon) \end{cases}
$$

$$
extract(H, \epsilon) = \begin{cases} H = \emptyset \vee H = [(\epsilon, A_\epsilon)|H'] \\ \quad \Rightarrow \emptyset \\ H = [(\epsilon', A_{\epsilon'})|H'], \epsilon), \epsilon' \neq \epsilon \\ \quad \Rightarrow [\epsilon'|extract(H', \epsilon)] \end{cases}
$$

$$
rewind(p, H, \epsilon) = (\, restore(p, H, \epsilon),\ extract(H, \epsilon)\,)
$$

Figure 4.12: Description of the $rewind$ function

In the example, after rewinding the process adaptations until just before the arrival of the event *fail*, we get the process depicted in FIG. 4.13

Figure 4.13: Rewound process before the *fail* event

4.3.4.3 M_3: Pruning the Adaptation History

The objective of this operation is to identify in a sequence of events the ones related to the to–be–removed event (immediately or transitively), and consequently reject them all (as they are now irrelevant). We define a $prune$ function to support this operation. Using as inputs the sequence of events computed by $rewind$ (named $History$) and a sequence of events to be rejected (named $Removed$, and initially containing the to–be–removed event), this function produces a $Pruned$ sequence of events. According to our objectives, the $Pruned$ sequence contains events that are not related to the ones defined in $Removed$. Its definition is represented in FIG. 4.14.

$$\begin{array}{rcl} prune : [\text{CE}] \times [\text{CE}] & \rightarrow & [\text{CE}] \\ (Hist, Removed) & \mapsto & Pruned \end{array}$$

$$prune(H, R) = \begin{cases} H = \emptyset \Rightarrow \emptyset \\ H = [\epsilon|H'] \wedge \exists \epsilon' \in R, \epsilon \in \epsilon' \\ \quad \Rightarrow prune(H', [\epsilon|R]) \\ H = [\epsilon|H'] \wedge \nexists \epsilon' \in R, \epsilon \in \epsilon' \\ \quad \Rightarrow [\epsilon|prune(H', R)] \end{cases}$$

Figure 4.14: Description of the $prune$ function

In the example, the list of events in the history is [*fail*, *perf*, *slow*, *cache*]. When we apply the prune of the event *fail* to the history, we get [*perf*, *slow*]. The event *cache* is removed as it depends on the previous existence of *fail*.

4.3.4.4 M_4: Replaying a Complex Event Sequence

The objective of this operation is to perform process re–adaptation, *i.e.,* to re–execute on the rewound process the adaptations that still need to be present in the expected result (*i.e.,* the adaptations triggered by the events identified by the $prune$ function). This operation is described in a function named $replay$, presented in FIG. 4.15. Using a given process p' and a sequence of events $[\epsilon_i, \ldots, \epsilon_j]$ as inputs, the function produces a process p_r that implements the expected result of the undo process.

$$\begin{aligned} replay : \mathcal{P} \times [\text{CE}] &\rightarrow \mathcal{P} \\ (p', [\epsilon_i, \ldots, \epsilon_j]) &\mapsto p_r \end{aligned}$$

$$replay(p, L) = \begin{cases} L = \emptyset \Rightarrow p \\ L = [\epsilon | L'] \\ \quad \Rightarrow replay(adapt(p, \epsilon), L') \end{cases}$$

Figure 4.15: Description of the $replay$ function

In the example, when replaying the pruned events on the process, we get the adaptations caused by the *perf* event (*i.e.*, monitoring of each activity). The event *slow* is ignored, as it is only important when preceded by the event *fail*. The resulting process is presented in FIG. 4.16, which is the expected result of a correct unadaptation.

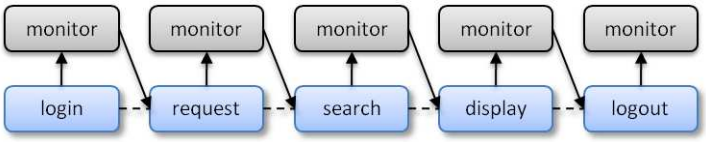


Figure 4.16: Correct unadaptation

It is important to note that *rewinding* and *replaying* all the events occurred between the reception of ϵ and $\neg\epsilon$ (and their corresponding adaptations), in the exact order in which they occurred, is essential for the correct undoing of the adaptation. The adaptations related to the events that were kept after the *prune*, which were undone during the *rewind* process, will usually be *replayed* in the same manner. However, after *pruning* the events we are creating a new set of events, which may result in a different set of adaptations.

4.4 Summary

In this chapter we presented our solution for doing and undoing dynamic business process adaptation, using context information and adaptation rules with an event-driven approach. We have argued that undoing business process adaptations is

an important issue that should not be obviated, since naive approaches could end up with corrupted processes. Moreover, we presented our solution for dealing with the challenge of **correctly undoing adaptations** in an event–driven approach, which included four mechanisms: *identify*, *rewind*, *prune* and *replay*.

Our approach is generic, since the detection of the undo trigger event is defined in terms of event processing engines, and uses boolean logic to associate an event to its non–event. Furthermore, our approach is also automated, since the way business processes are unadapted is fully delegated to an automatic engine. Thanks to these two advantages, we set the user free from having to deal with the unadaptation logic. Finally, our approach considers not only the original adaptation, but also the subsequent adaptations that were related to it, and unadapts them as well, while leaving the unrelated adaptations untouched, resulting on a cleanly unadapted process.

In the next chapter we will present the CEVICHE Framework, which is the SCA-based implementation of the work presented in this chapter. It uses its own adaptation language, called the ABPL, to specify adaptation rules and situations, and uses context information provided by CEP engines. Moreover, it uses a plug-in approach in order to be able to work with different implementations of CEP.

Chapter 5

The CEVICHE Framework

"If you want truly to understand something, try to change it."
- Kurt Lewin

Contents

5.1 Introduction

We are living among highly dynamic environments, where information is constantly flowing and the rules are continuously changing. These dynamic environments contrast with the current situation of business processes, where its static nature does not allow them to consider these continuous changes, and respond accordingly. The huge amount of data that could be taken into consideration cannot be easily specified using the standard business process languages, such as BPEL.

Given this background, we need a solution that can overcome the following challenges: ***i)* dynamic business process adaptation, *ii)* context integration,** and ***iii)* proper undoing of adaptations**. To achieve this, we created the CEVICHE Framework (*Complex EVent processIng for Context-adaptive processes in pervasive and Heterogeneous Environments*), which intends to add **simplicity, flexibility** and **dynamicity** to the business processes [Hermosillo et al., 2010b, Hermosillo et al., 2010a].

- **Simplicity** in the way that we separate the context-based decision making from the core of the process to an external entity, allowing the process to be focused only on the core business logic and not in other cross-cutting concerns.
- **Flexibility** because we intend to make every single step of the process to be a possible decision point, where the process can be adapted, so that it is not restricted in the kind of modifications that can be achieved to deliver the best response in a given situation.
- And **dynamicity** in the sense that all the adaptations to the process could be done at run-time, so that we do not lose any information and service time by having to stop and redeploy the business process with the desired changes in its behavior.

In the previous chapter we presented our formalized approach for dynamically adapting business processes using an event-driven approach. We also explained how undoing these adaptations is not a trivial task, and presented our proposal for correctly undoing adaptations. In this chapter we present our implementation of that solution, in the form of the CEVICHE Framework.

Structure of the Chapter

The rest of this chapter is organized as follows: Section 5.2 gives an introduction to events and presents an overview of how events are used for adaptation on the CEVICHE framework. Then, in Section 5.3 we explain the architecture of the CEVICHE framework. In Section 5.4 we present the global picture of the Adaptive Business Process Language. Section 5.5 presents the Adaptation Manager, in charge of dealing with all the adaptation information. Next, in Section 5.6 we show how the events are declared and the conditions defined, and we also explain the need for a plug-in approach. Finally, Section 5.7 summarizes the ideas presented in this Chapter.

5.2 Dynamic event-based adaptation

There has been some research towards the need of real-time processing of streams of data, which can help to obtain useful information from constantly changing environments. One of these approaches is *Complex Event Processing* (CEP), which considers every change in the environment as a simple event and helps the user to specify a set of rules that will be used to create higher level events, called complex events, that relate more to the business logic, allowing it to find the information that is important for a specific application. This information can then be used, for example, to adapt such applications in order to respond to the new conditions of the environment, by modifying their behavior.

CEVICHE is an event-based framework that intends to add flexibility to the static nature of existing business processes, by allowing them to be dynamically adapted in order to respond to changes in the context in which the business process is being executed. In this section we present how events are used in the CEVICHE Framework during the execution of the process, as well as an overview of the adaptation process using the context changes represented by those events with a component-based approach.

5.2.1 Events

Event management is a difficult task, as there is a continuous stream of events flowing from the sources, and each of the events needs to be analyzed and organized

in order to obtain the important information that they carry. CEP is an emerging approach for facilitating the management of events, from which business process managers can benefit, as it allows them to find real-time relationships between different events, using elements such as timing, causality, and membership in a stream of data to extract relevant information [Luckham, 2002]. In CEVICHE we use CEP to deal with our challenge of **context integration**, since it helps us to find the possible scenarios in which an adaptation might be needed, so that we can then relate those scenarios to specific and adequate responses. To achieve this integration, the CEVICHE framework follows a sequence of steps, which we depict in FIG. 5.1.

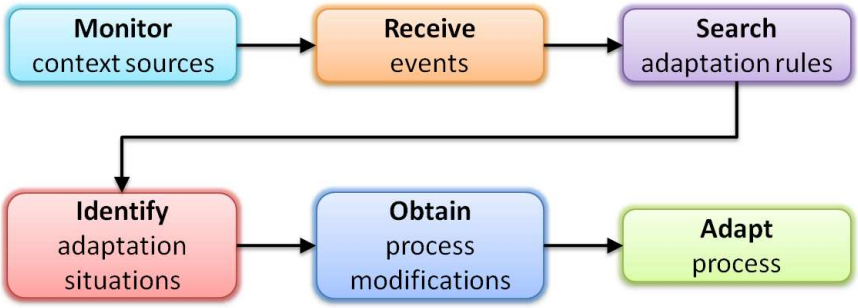


Figure 5.1: The adaptation sequence

The first thing to do, is to ***monitor*** the context sources, which can be any source providing information about the environment in which our process is running. Any change in the context is represented in the form of an event, and when they occur, they are ***received*** and processed by the CEP engine. The CEP engines need to be previously provided with the adaptation rules, which are then ***searched*** to see if the event corresponds to a situation that might trigger an adaptation. If the CEP engine finds a match, then it alerts the Adaptation Manager which in turn ***identifies*** the adaptation situation and then, from the rule repository, it ***obtains*** the modifications (*i.e.*, add or remove activities or links) that need to be done to correctly respond to the new context, which are then used to ***adapt*** the process.

5.2.2 Dynamic adaptation

During a business process execution, the events that are interesting to us are the ones related to changes in the context, which would affect the result or performance

of the execution. To monitor these changes we use CEP, which allows us to obtain meaningful information from different streams of data. By integrating this technology, we are able to create context-aware business processes that will consider the information around them during their execution, allowing the system to decide when and how the business process should be adapted, based on that information.

However, we still have to deal with the static nature of business processes, since in BPEL we cannot modify our processes at run-time. One of the challenges presented in this dissertation is the **dynamic adaptation of business processes**, which means that we need to add flexibility to the business processes by allowing them to be modified at run-time, without the need to redeploy the process after every change. To achieve this goal, we use a component-based approach, or more specifically, an SCA approach (Service Component Architecture) [Beisiegel et al., 2007]. SCA is a technology that aims to combine the advantages of Service-Oriented Architecture (SOA) with those of Component-Based Software Engineering (CBSE) for the development of SOA-based business applications [Kramer, 2008].

We transform the BPEL activities into components, using the same services defined in the WSDL description, binding the components according to the business process definition in BPEL. There are actually several similarities between a BPEL program and a component assembly. For example, both can be used to describe a business service that is implemented by composing together other business services and both can describe inbound and outbound service interactions types by WSDL port types [Edwards, 2007]. Using these similarities we can transform the BPEL process into components in a compatible manner.

For this transformation, variables in BPEL are transformed to SCA properties, basic activities (*e.g.*, `invoke`, `assign` or `reply`) are transformed as simple components, and structured activities (*e.g.*, `flow`, `sequence` or `while`) are transformed into composites, as they are activities containing other activities. In the case of partner links, these can either be services or references, according to the sense of the communication. In SCA, this sense is determined by the first message sent by one of the parties. So, the sender of the first message becomes the client and the receiver becomes the service provider, regardless of the number of messages sent and received by each party during the whole conversation. By doing an analysis of the control flow of the business process, we determine which part sends the first message and so we can establish if the role of the partner link is either a service or a reference. This is illustrated in TAB. 5.1.

BPEL	SCA
Variables	Properties
Basic activities (*e.g.*, invoke, assign, reply)	Simple components
Structured activities (*e.g.*, flow, sequence, while)	Composites
Partner links	Services or References (depends on communications sense)

Table 5.1: From BPEL to SCA

The dynamic adaptation of the process is achieved thanks to the reconfiguration capabilities provided by the FrasSCAti platform [Seinturier et al., 2009, Seinturier et al., 2012], as we have presented in Section 2.2.4. The bindings of the components can be modified during the execution of the process. Components can be added or removed from the architecture (reflected as adding or removing activities of the business process): this allows us to adapt the behavior of the business process to better respond to its current context, making it dynamically reconfigurable.

The context that is used to adapt the business process is monitored using CEP rules. These rules are defined by the user, using CEVICHE's adaptation language ABPL, and indicate the situations under which we may want to adapt the behavior of our business process. We will explain these rules later and the ABPL, in Section 5.4. When the context changes, an event is received and its associated adaptation is executed. This adaptation is done by adding and/or removing tasks from the business process, to adjust its behavior in order to respond better to the new context. The interaction among these different parties (ABPL, BPEL, SCA and CEP) is presented in FIG. 5.2.

Additionally, to improve the flexibility of our framework, CEVICHE intends to work with any CEP engine available. To achieve this, CEVICHE relies on a plug-in approach that allows it to translate from its own pivot language to the specific languages of the CEP engines. This approach helps the user to simplify the task of defining the events and adaptation rules, by doing it only once, instead of doing it for each different engine. In the following section we will present the architecture of our approach and later, in Sections 5.4 and 5.6, we will explain in more detail our pivot language, called the ABPL, and the use of the plug-ins.

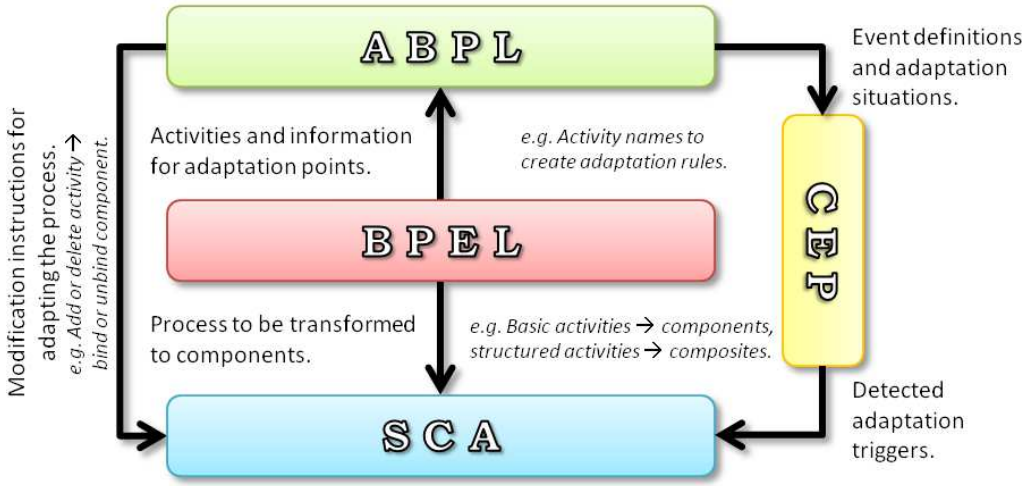


Figure 5.2: Relation among ABPL, BPEL, SCA and CEP

5.3 CEVICHE Architecture

In this section we present the architecture of the CEVICHE framework, and we give a brief description of each of the parts involved. To respond to the challenges presented during this dissertation, the architecture of CEVICHE is composed of four main parts: 1) the ABPL, where the user provides the event definitions and possible adaptations, sets the base for **context integration**; 2) a translation framework that separates the main business process rules from the adaptation conditions, 3) an adaptation manager that manages the alternative processes and deals with the process adaptation, which allows us to **dynamically adapt the business process**, and later to **properly undo those adaptations**; and 4) a translation plug-in in charge of adapting the event definitions for each *Complex Event Processing* (CEP) engine, which helps to tackle **CEP heterogeneity**.

At the same time, CEVICHE also relies on different technologies to achieve the process adaptation, like the CEP and BPEL engines, as shown in FIG. 5.3. For the BPEL engine, we use EasyBPEL[13], a WS-BPEL 2.0 engine that relies on EasyViper[14], which allows to build service-oriented workflows where the nodes of the execution graph are SCA components. As for the CEP engine(s), we are open to the use of any engine, as long as a plug-in for it already exists (as we explain in Section 5.6).

[13] http://research.petalslink.org/display/easybpel/
[14] http://research.petalslink.org/display/easyviper/

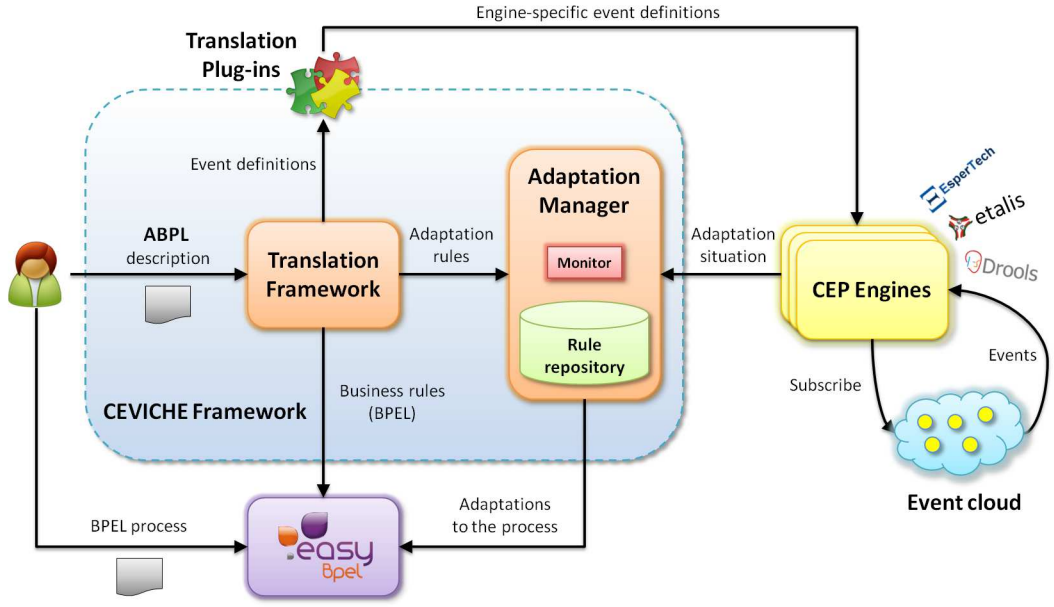


Figure 5.3: The CEVICHE framework

Adaptive Business Process Language

The *Adaptive Business Process Language* (ABPL) is used by the user to express how the business process is going to be adapted, as well as the situations that would trigger such adaptation. Here, the user defines the rules that will be used to adapt the process by specifying what activities will be concerned by the adaptation and whether new activities need to be added and where, or if some existing activities need to be removed. Those adaptation rules are triggered by the occurrence of events, which are also defined in the ABPL. The definition of these events will then be used by the CEP engines to monitor the context. The use of the ABPL is explained in more detail in Section 5.4.

Translation Framework

The *Translation Framework* is in charge of receiving the ABPL definitions from the user, and separating its contents to be processed by other parts of the framework. First, it separates the information concerning the adaptation rules and sends it to the Adaptation Manager, where they are processed and inserted into an adaptation

database, called a *rule repository*. The information concerning the event definitions is sent to the CEP engines, using a translation plug-in, where it will be used to monitor any changes in the execution environment. The role of the *Translation Framework* ends once the information on the ABPL definitions is sent, and does not have any active role during the execution or adaptation of the business process, other than updating the adaptation rules if new definitions are provided.

Adaptation Manager

This is one of the most important parts of the CEVICHE framework. As its name suggests, it deals with the adaptation logic of the business process, and is the core of our framework for dealing with **dynamic process adaptation** and **correct undoing of adaptations**. Using the information specified in the ABPL, the *Adaptation Manager* creates a relation between events reported by the CEP engine and the adaptation rules. It contains a light database (*rule repository*) that stores these relations and that is used to know when an adaptation has been generated, at what moment, and under which circumstances. The order in which adaptations occur is very important, since it is needed later when we want to undo the adaptations. Undoing adaptations is not a simple nor straight-forward task, and the logic to achieve it is also handled by the adaptation manager. A more detailed explanation about how the adaptations are managed will be given in Section 5.5.

Translation Plug-ins

The *Translation Plug-ins* are the only interchangeable parts of the CEVICHE framework. Their main goal is to help CEVICHE to handle **CEP heterogeneity**. Given the diversity that exists in the CEP implementations, it is not possible to specify the event definitions in one engine and then use that specification in another one. With CEVICHE, we use the ABPL as a pivot language, where the event definitions are only provided once, and then they are translated to the language of the engine chosen by the user. The use of the *Translation plug-in*, as well as some examples of the result are explained later, in Section 5.6.

5.4 Adaptive Business Process Language

We have previously discussed of how CEVICHE intends to improve **context integration** in business processes and for that, it uses CEP. However, there is no actual integration between CEP and BPEL. When defining our process execution in BPEL, we cannot easily specify a complex set of conditions to be evaluated on-the-fly and then respond accordingly. We could certainly define a limited number of conditions, and the activities to execute after them, but we would need to know beforehand all the possible situations that we might face. Moreover, this would create an unmaintainable, spaghetti-like set of conditions that would need to be constantly updated as new scenarios present themselves.

In order to respond to the lack of adaptation specifications in the current standards, we created the *Adaptive Business Process Language* (ABPL). The goal of the ABPL is to work as a pivot language, where users can express the context information they want to monitor, in the form of events, as well as the adaptation conditions and actions to respond to those changes. More than extending BPEL, the ABPL works as an add-on, since it uses the data from the main process definition in BPEL, but does not modify the original information, which makes it more transparent.

This approach also helps to maintain a better separation of concerns, since we can leave only the core business logic in the BPEL process, without including all the specific situations. Our goal with this approach is to facilitate the integration of CEP with existing business processes and to make the use of this technology easier, without the drawbacks of early adoption. As an adaptation language, in ABPL we answer the four basic adaptation questions: ***What*** to adapt?, ***When*** to adapt it?, ***Where*** to adapt?, and ***How*** to adapt it? [McKinley et al., 2004b, McKinley et al., 2004a]. These questions are discussed in more detail in Section 5.4.2. The Document Type Definition (DTD) of the ABPL can be seen in FIG. 5.4. The different concepts that it introduces are described in the next subsections.

5.4.1 Adaptation and context integration with ABPL

An ABPL definition is mainly composed of two parts. The first part is related to **context integration**, and contains the description of the events. It is used to express

```
<?xml version="1.0"?>
<!DOCTYPE abpl[
  <!ELEMENT events (event+)>
  <!ELEMENT adaptation (situation+)>
  <!ELEMENT event (condition,attribute*)>
    <!ATTLIST event name CDATA #REQUIRED>
  <!ELEMENT condition (#PCDATA)>
  <!ELEMENT attribute EMPTY>
    <!ATTLIST attribute name CDATA #REQUIRED>
    <!ATTLIST attribute value CDATA #REQUIRED>
  <!ELEMENT situation (situation-event+,process*)>
    <!ATTLIST situation name CDATA #REQUIRED>
    <!ATTLIST situation idempotent CDATA #IMPLIED "true">
  <!ELEMENT situation-event (#PCDATA)>
  <!ELEMENT process (activity*,adapt)>
    <!ATTLIST process name CDATA #REQUIRED>
  <!ELEMENT activity (#PCDATA)>
  <!ELEMENT adapt (add?,del?)>
  <!ELEMENT add (activity*,link*)>
  <!ELEMENT link EMPTY>
    <!ATTLIST link from CDATA #REQUIRED>
    <!ATTLIST link to CDATA #REQUIRED>
  <!ELEMENT del (activity*,link*)>
]>
```

Figure 5.4: The ABPL DTD

the complex events that will be sent to the CEP engine. The second part specifies the details of the **dynamic process adaptation**, including under which circumstances should the business process be adapted and how to make such adaptation (add or remove activities or links). To simplify and facilitate the explanation and understanding of the different parts, we will use a simple ABPL descriptor as a reference, which can be seen in FIG. 5.5.

When declaring an event, we are actually creating a complex event based on conditions related to the occurrence of the combination of other simple or complex events. The absence of an occurrence within a window of time, can also be considered as a condition. Events can come in different forms and from multiple sources. For instance, we can have low-level events coming from sensors in the environment (*e.g.*, cpu usage, temperature changes, etc.), or high-level events coming from the business process (*e.g.*, product in warehouse, delivery started, etc.), or complex events coming from the CEP engine.

```
1  <abpl>
2    <events>
3      <event name='eventName'>
4        <condition>
5          cond1
6          ... [and|or]
7          condN
8        </condition>
9        <attribute name='attribName' value='attribValue'/>
10     </event>
11   </events>
12
13   <adaptation>
14     <situation name='situationName'>
15       <situation-event>eventName</situation-event>
16       <process name='processName'>
17         <activity>[activityName|ALL]</activity>
18         <adapt>
19           <add>
20             <activity>activityName</activity>
21             <link from='activityName' to='activityName'/>
22           </add>
23           <del>
24             <activity>activityName</activity>
25             <link from='activityName' to='activityName'/>
26           </del>
27         <adapt>
28       </process>
29     </situation>
30   </adaptation>
31 </abpl>
```

Figure 5.5: An ABPL descriptor

To define an event, the first thing we need to do is specify a unique name for it (line 3 of the reference structure), since this name acts as an id that can later be referenced in other conditions to create new events. Then we can define the conditions under which this event should be created (lines 4-8). The evaluation of these conditions can either be done by considering only the existence of another event, or by considering also the values of its attributes.

Our created events can also have attributes of their own, which are optional. We can assign static values to the event's attribute, or use others event's attributes as input (line 9). These attributes can then be used by other rules. If the combination of the conditions (lines 4-8) is met, the new event is created and published. In Section 5.6 we describe in more detail how the conditions of the events can be

created, as well as their syntax, and how can we assign static and dynamic values to the attributes.

5.4.2 An Adaptation Language

The ABPL is an adaptation language, and as we have previously stated, we need to answer four basic questions with it: ***What*** to adapt?, ***When*** to adapt it?, ***Where*** to adapt?, and ***How*** to adapt it? [McKinley et al., 2004b, McKinley et al., 2004a].

What to adapt? To answer the first adaptation question, we need to identify *what* we want to adapt. In this case, we want to adapt our business process, and since there may be several processes running, we need to identify the business process that will be adapted by its name (line 16).

When to adapt? With the business process identified, we then need to know *when* to adapt it, and this is specified using the `situation` and `situation-event` tags (lines 14 and 15). It is important to notice that by default, the adaptation situations are taken as idempotent, which means that they will only be considered once when adapting the business process. This helps to avoid re-adapting the process over and over for the same situation if it has already been done. For example, if we want to add an activity "α" when the situation described by a complex event "ϵ" arrives, then if we receive ϵ n times, we would end up having n times α. If a specific situation requires this kind of behavior, it can be specified as an attribute of the situation definition.

Where to adapt? The next question is *where* should we adapt the business process. This is specified with the `activity` tag (line 17), which acts as an adaptation point. The activity or activities specified in this part will be considered as the departing point of the adaptation. Though this tag is optional, it can be very useful when targeting several adaptation points in the process. For example, if we want to adapt two different places of the process in the same way, we only need to specify one adaptation sequence and apply it to the referenced activities. In this case, the keyword `ALL` may also be used to specify that all the activities in the process are concerned (*e.g.*, to add monitoring information).

How to adapt? Finally, to specify *how* the process is adapted, we define a sequence of modifications, adding and/or deleting activities and links (lines 18-27). In this moment, we can reference the adaptation point activity (line 17) by using the keyword `THIS`. This is specially useful when the adaptation is made in several points of the business process. Also, the `activityName` inside the `adapt` tag can either be a single activity, or a group of activities (*i.e.*, a sequence or a flow). To prevent problems concerning the deletion of an activity that is currently being executed, when deleting activities in an adaptation, the only thing that is really removed is the incoming link to it. Doing this, if an activity is currently being executed, the current process will be able to continue its normal execution, while a new process would not be able to access it, after the adaptation.

In FIG. 5.6 we can see how the adaptation rules are formed, using the information from different sources. First, the event definitions are used to create the complex events in the CEP engine, while at the same time they can be used to create new events and adaptation rules. Then, from the business process we can obtain the activities that will serve as the adaptation points for the rules. Finally, we can use the additional activities provided by the user to adapt the business process when necessary.

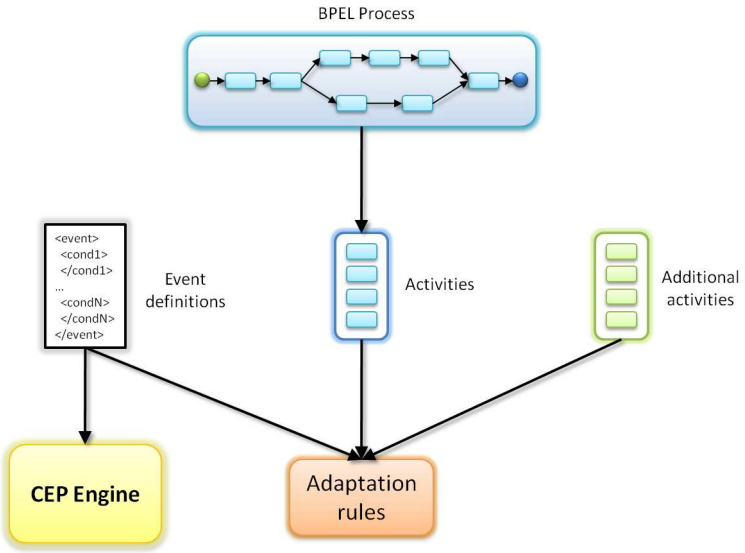


Figure 5.6: The adaptation rules

By separating the event definitions from the business process definitions, we can increase the flexibility of the process adaptation, since the conditions of the event definitions can be easily managed, inserting, updating and deleting events without affecting the business process. This also helps to foster the separation of concerns, avoiding to mix the decision making process with the business process definition.

5.5 Adaptation Manager

To deal with all the adaptation information and rules, we have an Adaptation Manager. This is one of the main components of the CEVICHE architecture, as it handles all the logic for doing and undoing changes to the business process. The Adaptation Manager receives all the adaptation information from the rules provided by the user using ABPL, and stores them in its *rule repository*, as is shown in FIG. 5.7. Those rules specify all the actions that need to be taken for the adaptations to take place (*i.e.*, adding or removing activities or links), as well as the specific situations that will trigger the execution of those adaptations, as we previously presented in Section 5.4.

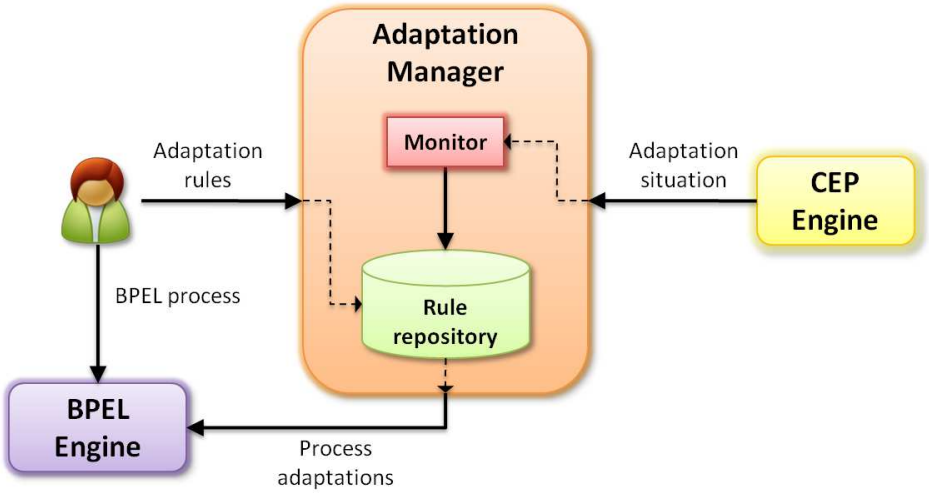


Figure 5.7: CEVICHE Adaptation Manager

To determine when an adaptation is to be executed, we need to be able to monitor the context and find the specific situations that will trigger it, and for this we use CEP. On the right-hand side of the figure we can see that the Adaptation Manager receives the information about the events that are related to the adaptation, which are obtained by subscribing a *monitor* to the CEP engine. We use a rule repository, to index the adaptation rules according to the adaptation situations that trigger them, so that it is easier to detect which adaptations are going to be applied to the business process at the time when a specific event arrives.

When an adaptation is applied, it is marked in the repository as used, unless it is explicitly defined as non-idempotent by the user in the adaptation definition (FIG. 5.4). This will prevent the Adaptation Manager from applying the same adaptation over and over when receiving an event more than once. For example, if we want to add a monitoring activity "α" when the situation described by a complex event "ϵ" arrives, then if we receive ϵ n times, we would end up having n times α.

Another task of the Adaptation Manager is to keep track of all the events that trigger each adaptation, and of all the adaptation steps that are taken each time. The goal of this is to have, on one hand, a log that will let us to know how the business process reached its actual state, and on the other hand, a history that will allow us to correctly undo the adaptation process, when those adaptations are no longer needed.

Undoing adaptations is usually considered a trivial task, but as we explained in Section 4.3, this is not the case and we need to be specially careful when dealing with cumulative adaptations (*i.e.,* adaptations that can be applied over previous adaptations). To correctly achieve adaptation undoing, the Adaptation Manager detects when the trigger of an adaptation is no longer valid, by detecting its opposite event, as explained in Section 4.3.4. Once detected, it starts the undo process (*i.e., rewind*, *prune*, *replay*) using the history of events and adaptations that were logged.

5.6 Translation Plug-ins

As we presented in section 5.4, event definitions are one of the main parts of the ABPL. However, in order to be able to interact with multiple engines capable of processing the information in those events, CEVICHE uses a plug-in approach which allows to transform the general definition in ABPL to the specific language of each

engine. In this section we will present how to create an event definition, and then use the plug-ins translate it to different engines.

5.6.1 Specifying Events with ABPL

In CEVICHE, events are considered as a source of context information. When using the ABPL, events are identified by the keyword `event`. We can reference the occurrence of an event by using the structure `event`[*'name'*]. If we want to access the value of an attribute that belongs to an event, we would then use the structure `event`[*'name'*][*'attribute'*].

With the notations of complex event definitions (conjunction, disjunction, sequence, etc.) described in TAB. 4.2 (p. 67), we can use occurrences and attributes to specify the different conditions that are needed to create new complex events. These new events can be empty (as a simple occurrence), or they can also contain their own attributes. The value of these attributes can be either static or dynamic. A static attribute value is assigned as a normal string attached to the attribute, while a dynamic value involves some pre-processing before being assigned.

Dynamic values are also divided in two types: event related values, and engine-specific function values. The former represents information that can be obtained from simple operations (*e.g.,* aritmetical operations), or that reference the attributes of the events involved in the condition. These type of values are identified with the prefix `@eval()`. The latter is used when instructions need to be sent to an event processing engine *"as is"*, which means that they will not be translated by the plug-in. This would help to cover the cases where a special functionality of the engine needs to be exploited, which is not implemented in all other engines. These type of values are identified with the prefix `@system()`.

An example of an event definition is shown in FIG. 5.8. Here we define an event called `Overload`, which is triggered whenever an `Error` event is received, and the type of error is either 500 or 503. There must also be a CPU load above 80% for the `Overload` event to be triggered. When these conditions are met, the new event is sent, containing as an attribute the kind of error that generated the overload, which is obtained from the `Error` event.

```
<event name='Overload'>
  <condition>
    event['Error']
    and (event['Error']['type'] = '500'
    or event['Error']['type'] = '503')
    and event['Load']['cpu'] > '80'
  </condition>
  <attribute name='error' value=@eval(event['Error']['type'])/>
</event>
```

Figure 5.8: Event definition example

5.6.2 A plug-in approach

One problem about using CEP is that, as useful and interesting as it is, the engines that implement this technology are still in an early phase, and as such they are prone to constant changes. Several engines have been created, some of them have merged, other simply absorbed by bigger competitors. As some important industry players such as IBM, Microsoft, Oracle and TIBCO enter the game, the light of stability begins to shine. Nevertheless, as an early technology, there is still no standard in the way in which the user-defined rules should be specified, and the language used in one engine may change completely from the one used in another.

This means that if we start using a CEP engine and then decide to change, because our engine is no longer being maintained, etc., we would need to specify all the rules yet again, so they can work in the new engine. The idea of CEP has been quickly spreading, and every time more and more enthusiasts are looking forward to use it, creating different implementations of engine and their own languages to express the rules. Since there is no standard as of how to define the events in the engines, the way in which the user needs to express the rules varies from implementation to implementation.

The language style varies from stream-oriented implementations (Esper[15], MS StreamInsight[16]), to inference rules (Drools[17], Tibco BusinessEvents[18]), to even logic programming (Etalis[19], Prova[20]). But even within the same language style,

[15] http://esper.codehaus.org
[16] http://msdn.microsoft.com/en-us/library/ee362541.aspx
[17] http://www.jboss.org/drools
[18] http://www.tibco.com/products/business-optimization/complex-event-processing/businessevents
[19] http://code.google.com/p/etalis/
[20] http://prova.ws/index.html

there are still slight differences that make each implementation different. Moreover, as the market continues to grow, the bigger companies are absorbing the smaller ones, while others are merging to create more competitive alternatives, or simply disappearing due to lack of support. This results in certain instability that refrains some people from adopting the technology, since there is no generic approach that can be evolved and reused independently from the constant changes of the solutions.

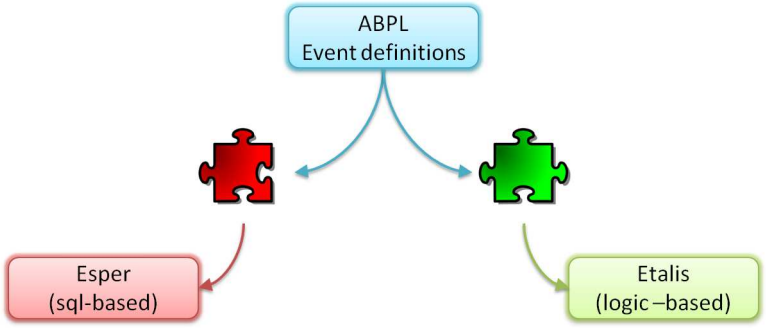


Figure 5.9: The adaptation plug-in

To face this problem, we use a plug-in approach that facilitates the use of CEP, without the drawbacks of early adoption. Using the ABPL as a pivot language, we are able to translate the user-defined rules and events to be used by any other CEP engine. To achieve this, we need to create a different plug-in for each CEP engine, as can be seen in FIG. 5.9. The advantage of this approach is that, even though this may seem as an exhaustive and complicated task, it only needs to be done once. And not only once for each user or set of rules, but only once for every implementation of a CEP engine, which means that once a plug-in has been developed, it can be shared by all the people using the same CEP engine.

Another advantage of this approach is that, if at any moment a standard is defined by the *Event Processing Technical Society*[21], there could just be a plug-in to the new standard and we could still be able to use our previously defined rules with any standard compliant engine, without the need to re-write them.

Also, as most of the CEP implementations can be grouped in just a few categories of language styles, once a plug-in for that style has been done, we can use it as the base for another similar language (*e.g.*, in a DB environment we could go

[21] http://www.ep-ts.com

from MySQL to Oracle with just a few minor adjustments). As a proof of concept we developed two basic plug-ins for two different open-source CEP engines: Esper and Etalis. In FIG. 5.10 we can see how the `Overload` event of FIG. 5.8 is translated to the SQL-like language of Esper, while in FIG. 5.11 we can see the translation to the logic-based language of Etalis.

```
insert into Overload(error)
select a.type
from pattern[
  every a=Error -> (type = '500' or type = '503')
  and Load.cpu > '80'
];
```

Figure 5.10: Esper 'Overload' complex event example

```
Overload(error) <- ( Error(type) and Load(cpu) )
  where ( error=type,
    ( =(type,'500') or =(type,'503') )
    and >(cpu,'80')  )
print_trigger(Overload/1).
```

Figure 5.11: Etalis 'Overload' complex event example

5.7 Summary

In this chapter we presented CEVICHE, a framework to create context-aware and dynamically adaptable business processes. We showed how, using CEP, we could monitor the context for our business processes, and how with a CBSE approach we are also able to adapt our processes dynamically. We introduced the architecture of the CEVICHE framework, and its four main components (ABPL, Translation Framework, Translation Plug-in and Adaptation Manager). We have presented the Adaptive Business Process Language, which intends to facilitate the adoption of CEP technology to be used in the creation of dynamic business processes. We explained the composition and structure of the language, and described in more depth its two main parts: events and adaptations.

For the adaptations, we showed why we may consider the ABPL as an adaptation language and how we could adapt the business process by adding and/or removing activities and links. As for the events, we have presented how complex events can be defined using the ABPL, and the rules to express the conditions that would eventually trigger a new event. Finally, we described why our plug-in approach can help to the early adoption of CEP by adding stability and flexibility to the users, and presented some examples of how one event definition is translated into two different implementation languages.

We presented a solution to our main challenges: **dynamic business process adaptations** and **context integration**, as well as to **CEP heterogeneity**. With this chapter we conclude the contribution of this dissertation, which consisted in the presentation of the CEVICHE framework with the ABPL to define adaptation events and situations and a description of how the adaptations are done and undone in our approach. In the following chapter we will present the validation of our work, and we will show the implementation of a nuclear crisis management scenario to demonstrate the qualities of our solution.

Part III

Validation

Chapter 6

Validation

"If the only tool you have is a hammer, you tend to see every problem as a nail."
- Abraham Maslow

Contents

6.1 Introduction

In this Chapter we present a scenario that integrates event management and business process adaptation, to be used as a case study and validation of our proposal. The goal of this scenario is to highlight the main challenges of our work: ***i)* dynamic business process adaptation, *ii)* context integration,** and ***iii)* proper undoing of adaptations**, as well as the sub-challenge of dealing with **CEP heterogeneity**.

We evaluate our approach by comparing the cost in time of changing the business process by redeploying it against the time it takes to adapt it dynamically. We also consider the overhead that adding dynamic adaptation brings to the execution of the business process, and show that when executed under normal circumstances, the overhead is negligible. Finally we analyze the cost of undoing adaptations, and show how this cost varies depending on the number of subsequent adaptations that happened after the one we want to undo.

Structure of the Chapter

The rest of this chapter is organized as follows: In Section 6.2 we present the case study that we used as validation for our work. Section 6.3 shows the implementation of some of the rules and event definitions of the scenario. Next, Section 6.4 presents a quantitative analysis of our approach. Finally, we conclude in Section 6.5 with a summary of the ideas presented in this Chapter.

6.2 Case Study: Nuclear Crisis Management

In this section we present a nuclear crisis management scenario[22] that has been simplified for easier comprehension. In these kind of scenarios, several actors are involved, and each of them play an important role in the successful execution of the process. At the same time, there are several complex procedures which need to be automatized and defined precisely to assist the correct operation of the rescue teams. There is also a strong need for reactivity in order to take into account the diverse context changes that would present new scenarios to deal with.

[22]This scenario was conceived before the sad events and crisis that happened on Fukushima, Japan.

Our scenario is based on the information and response actions provided by the French Nuclear Safety Authority[23] (*Autorité de Sûreté Nucléaire*). We also considered different situations from multiple real simulations [24] that have been done in France to test the reactivity of the different stakeholders that need to collaborate in order to solve the crisis.

6.2.1 Description of the scenario

A `nuclear power plant`[25] is operating normally near a residential area, when suddenly a problem occurs with the ventilation system, which damages the isolation inside the nuclear plant, creating a leak of radioactive steam to the environment. The `Radiation Survey Network` detects high levels of radioactivity coming from the contaminated steam. This generates a nuclear crisis alert, and the `Prefect` of the area is contacted to start with the emergency procedures.

After confirming the event, the `Prefect` officially declares a nuclear crisis, and begins the crisis management process. The first step is to establish the Emergency Operations Center (`EOC`), which is formed by the head or representative authorities of the different entities that will be involved during the containment of the problem (police, firemen, Emergency Medical Services (EMS), etc.). A `spokesman` is assigned to inform the media about the crisis, and to future developments of the situation. The media plays an important (though external) role as the channel to communicate with the people, and to extend any advices or alerts from the `EOC`. Buses are requested from public and private services to be used as evacuation vehicles for the people around the crisis area.

Three supporting parties are contacted to provide information that will be used to know how to proceed during the crisis management: the radiation survey network, a weather station and a scientific cell. The `radiation survey network` monitors and provides information about the radiation levels around the crisis area. The `weather station` provides information about changes in the weather that can extend the possibility of contamination to other areas (*e.g.*, strong wind, heave rain, etc.). The `scientific cell` is a group of experts that give advice to

[23] http://www.asn.fr/

[24] http://www.dissident-media.org/infonucleaire/news_simu_ac.html

[25] The words in the scenario description shown in `typewriter` font, represent the actors of the process.

the EOC about how to proceed to better contain the situation. They use the information provided by the other supporting parties, as well as the information about the current state of the crisis provided by the EOC.

The scientific cell advices the EOC to create a safety perimeter of at least 10 km and to install iodine distribution posts to help diminish the possibility of the people getting radioactive poisoning. The spokesman informs the people and the media about the evacuation points and buses, as well as the iodine distribution posts, and advices the people to consume the iodine pills as soon as possible.

The police arrives at the scene and receives the order to create a safety perimeter of 10 km around the crisis area. They start deviating all incoming traffic to maintain the perimeter. The Red Cross installs medical posts around the crisis area to receive any victims of the radioactive effects. The firemen, supported by the army, begin to evacuate all the people inside the safety perimeter and take them to the evacuation points where the buses will pick them up.

At the same time, the firemen search for any victims that may need assistance. When victims are located, they contact the Emergency Medical Services (EMS) so that the victims can be transported to the medical posts, where they are treated and monitored by the physicians. After transporting contaminated victims, the units of the EMS are brought to the decontamination center, where the vehicles and personnel are cleaned from any radioactive exposure, to prevent spreading any contamination.

Later, a smoke detector located inside the nuclear plant gets activated, meaning a fire has begun in the plant. The EOC decides to sound the evacuation alarm and report an explosion risk. The firemen are sent to the place to fight the fire, and the army is requested to evaluate the explosion risk. The safety perimeter is advised to be extended to 20 km by the scientific cell, and all the actors are notified about the risk. A new traffic plan is created and enforced by the police, along with the new safety perimeter.

The weather station reports strong winds of over 25 km/h coming from the east. The scientific cell notifies the EOC about the risk of radioactive particles being carried by the wind and advices to move all the established posts (medical and distribution) on the west side of the accident farther away. The EOC gives the order to move the posts to farther locations. The police is also notified,

and they start deviating the traffic on the west side of the area. When the fire is controlled, the `army` decree that the explosion risk no longer exists and the extensive measures are taken back by the `EOC` (*i.e.*, the `firemen`, `army` and `police` return to their previous activities).

6.2.2 Roles of the scenario

The presented scenario contains several actors participating in different situations, which are denoted in the description by using `typewriter` front. Each of those actors has specific responsibilities that need to be covered and a defined number of actions that she/he can do. These actions and responsibilities may change to adapt to a new situation whenever there is a change in the context. To help structure the business process definition of the scenario, we divided the participating actors into four groups that represent their role in the scenario: *Decision making*, *Operational*, *Supporting*, and *Consulting*, as is presented in FIG. 6.1.

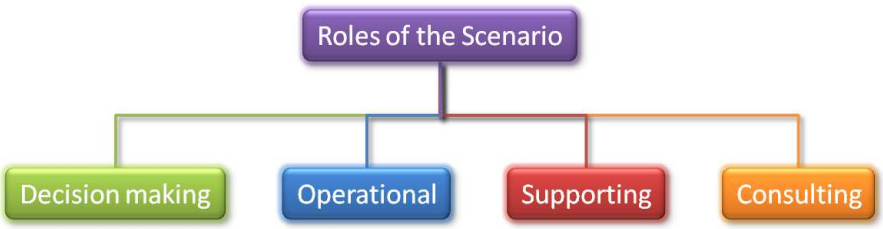


Figure 6.1: Roles of the Scenario

Decision making role. The actors with this role are the ones in charge of evaluating all the information surrounding the situation, analyzing the possible solutions and deciding the better way to proceed. In the scenario, this role is taken by two entities: the `Prefect` and the Emergency Operations Center (`EOC`, as presented in FIG. 6.2. These actors are the heads of the whole process and are only there to manage the situation. They do not interact directly with the crisis site, however, any situation that may require a change in the normal execution of the problem has to be analyzed and decided by them.

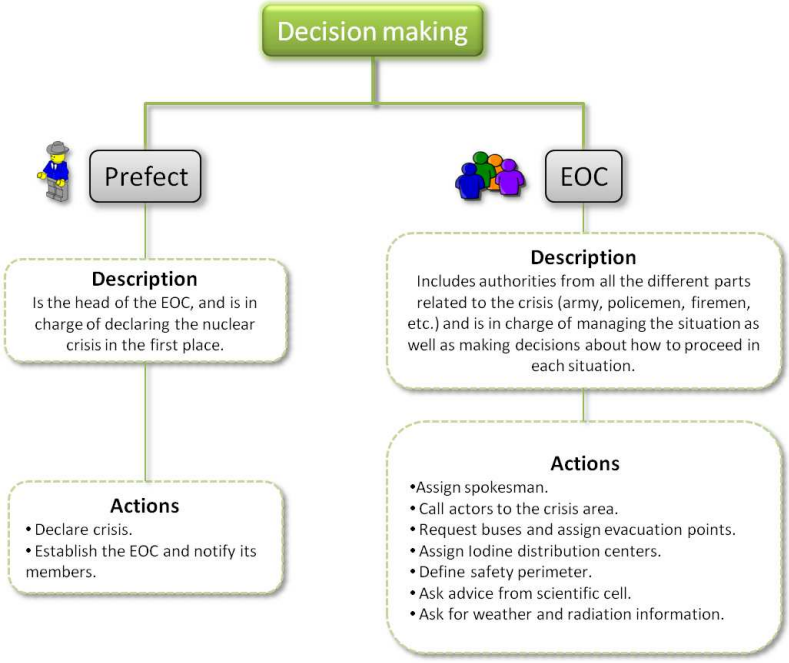


Figure 6.2: Decision Making Roles

Operational role. The actors with this role are the ones that interact directly with the crisis site. They execute any orders received by the *decision making* roles. In the scenario, this role is taken by four entities: the `Police`, the `Firemen`, the Emergency Medical Services (`EMS`) and the `Physicians`, as presented in FIG. 6.3. Even though they have a preestablished sequence of tasks to execute, their behavior can be modified by a *decision making* actor if the situation changes.

Supporting role. The actors with this role are the ones that help in specific situations and activities, related to the main goal of the process, which is the management of a nuclear crisis. They may interact with the crisis site, but only to support the tasks of other actors with *operational roles*. In the scenario, this role is taken by three entities: the `Army`, the `Red Cross` and the `Spokesman`, as can be seen in FIG. 6.4. In case the situation evolves, the actors with this role can take a more active part of the process by becoming *operative* actors to respond to the new situation, following a given order by a *decision making* actor.

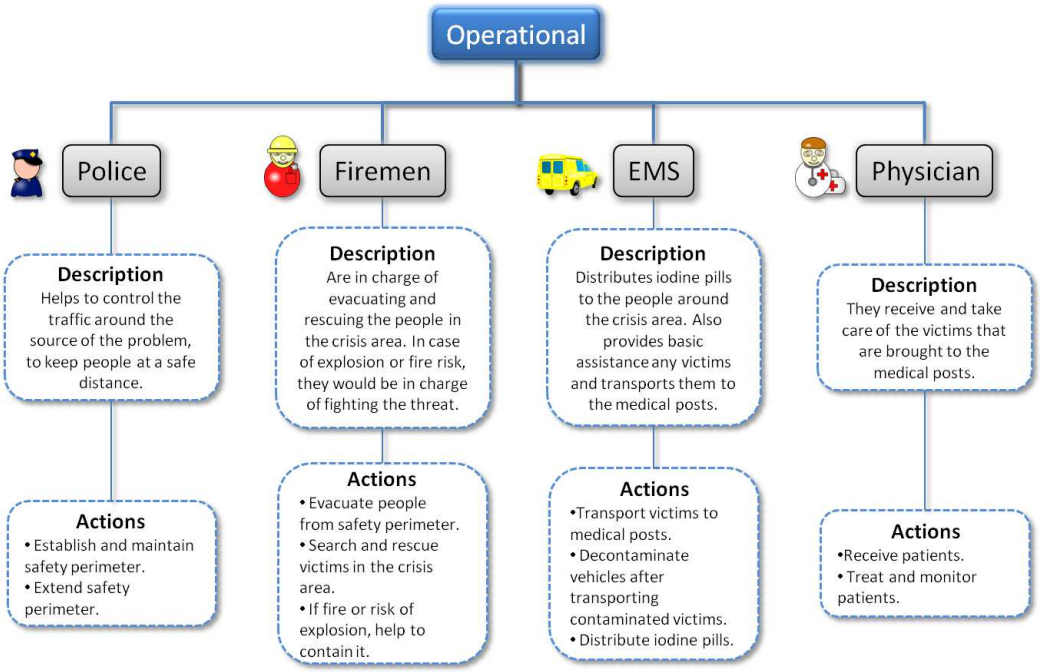


Figure 6.3: Operational Roles

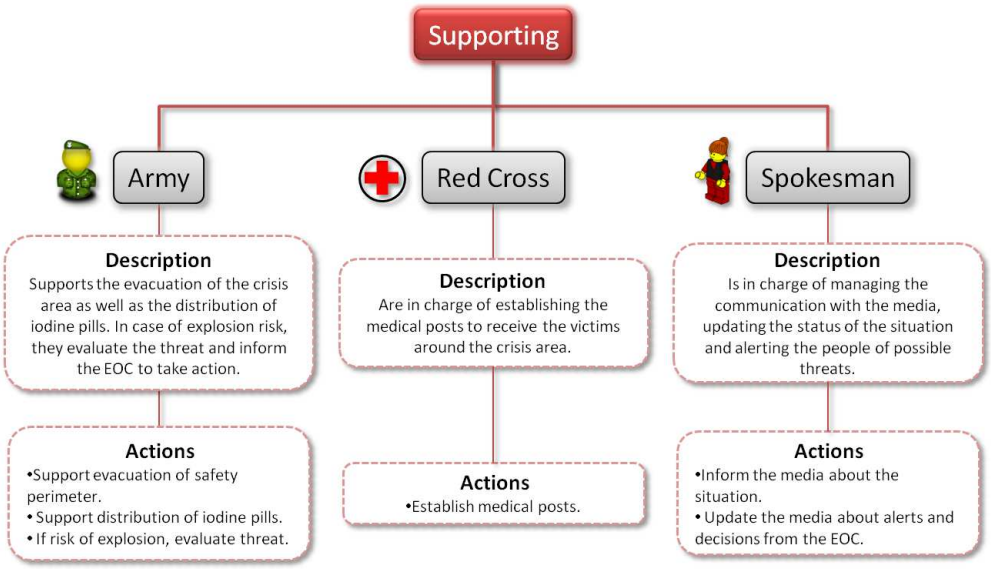


Figure 6.4: Supporting Roles

Consulting role. The actors with this role are completely alienated from the crisis site, and only provide information about the situation or external factors that may have implications on it. In the scenario, this role is taken by three entities: the `Scientific Cell`, the `Radiation Survey Network` and the `Weather Station`, as is presented in FIG. 6.5. Even though they do not have an active part in the management of the crisis, this is a very important role, as the information provided by the actors with this role is crucial for the successful accomplishment of the process, and it can also lead to adaptations of the original process given the changes in the context.

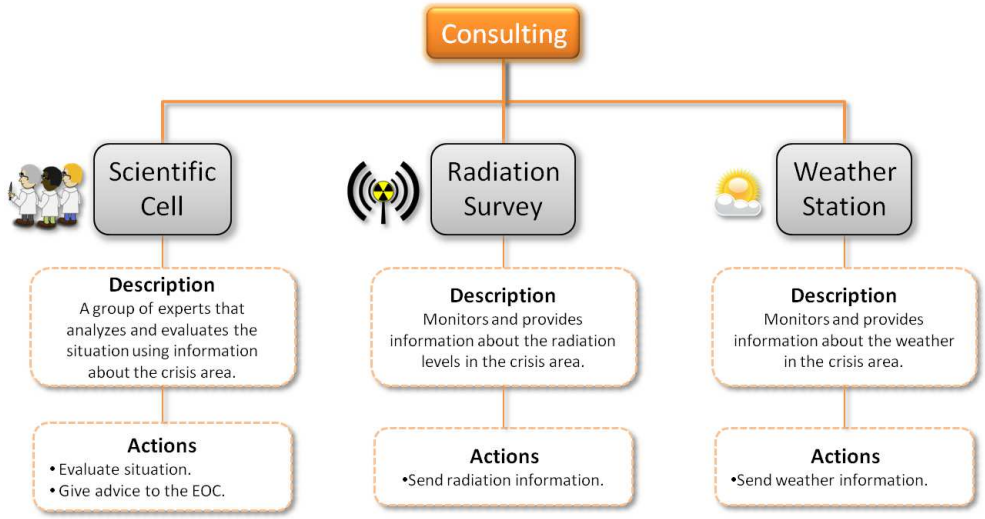


Figure 6.5: Consulting Roles

Finally, in the scenario we can find different places that are key during the development of the crisis management, as they interact with the *operational* actors during the process. These places are described in FIG. 6.6, and include: the `Nuclear Plant`, the `Medical Post`, the `Decontamination center` and the `Iodine Distribution Center`.

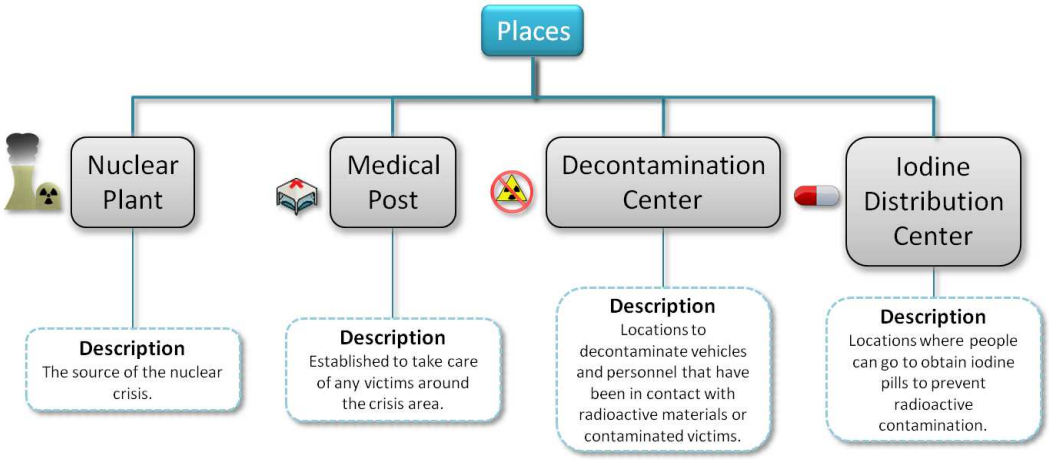


Figure 6.6: Places of the scenario

6.3 Implementation and Qualitative analysis

In the scenario presented before, we can identify one main process that is started by the `Prefect` with the goal of managing the nuclear crisis. This process is depicted in FIG. 6.7. However, during the development of the scenario we can also notice that external circumstances (such as the strong winds and later the risk of an explosion) cause a change in the context which affects the execution of the main process. Thanks to the dynamic reconfiguration capabilities provided by SCA (in the form of the FraSCAti platform), we can handle **dynamic business process adaptation** on the CEVICHE framework.

To help the separation of concerns, as well as to improve the maintainability of the process, we model the response to the external circumstances as adaptations to the main process. In case they are needed, they are automatically integrated with the main process. The changes in the context of the process execution can be determined by monitoring the information provided by different sources (*e.g.*, the `radiation survey network` or the `weather station`). Whenever the context information changes, an event containing the new information is received. Events may contain several attributes, and the events used in this scenario are presented in TAB. 6.1.

Using this event-based approach allows CEVICHE to foster the **integration of the context** with the business process. We define the way in which these events

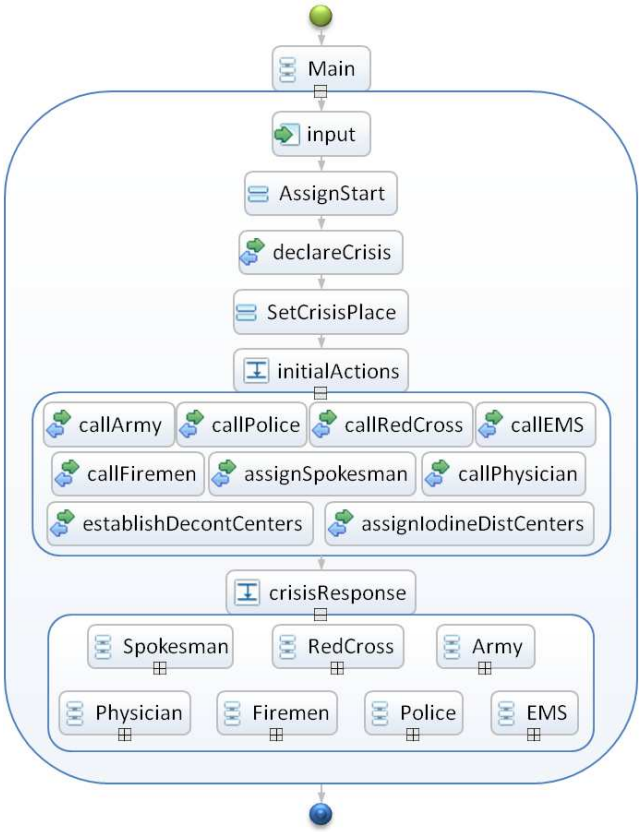


Figure 6.7: Main process of the scenario

Event	Attribute	Type
Weather	temperature	°C
	windDirection	[N | W | S | E | NNW | NW | ...]
	windSpeed	km/h
	rain	mm
	snow	mm
Radiation	location	text
	level	mSv
SmokeDetector	smokeDetected	[TRUE | FALSE]
	temperature	°C
Expert	situation	text
	advice	text

Table 6.1: Events of the scenario

are to be interpreted using ABPL to express the desired criteria, which will later be translated to CEP rules. In FIG. 6.8 we show the ABPL description of some of these events, that will be transformed to CEP rules. These events generate new events, which may contain new attributes. At the same time, these attributes can be assigned to new values or we can assign the values contained in the detected events, using the `@eval` call to obtain them.

```
<event name='radiationAlert'>
  <condition>
    event['Radiation']['level'] >= '20'
  </condition>
  <attribute name='level' value=@eval(event['Radiation']['level'])/>
  <attribute name='location' value=@eval(event['Radiation']['location'])/>
  <attribute name='time' value=@system('timeStamp.toString()')/>
</event>
<event name='strongWindAlert'>
  <condition>
    event['Weather']['windSpeed'] >= '25'
  </condition>
  <attribute name='speed' value=@eval(event['Weather']['windSpeed'])/>
  <attribute name='direction' value=@eval(event['Weather']['windDirection'])/>
</event>
<event name='heavyPrecipitationAlert'>
  <condition>
    event['Weather']['rain'] >= '25'
    or event['Weather']['snow'] >= '75'
  </condition>
  <attribute name='rain' value=@eval(event['Weather']['rain'])/>
  <attribute name='snow' value=@eval(event['Weather']['snow'])/>
</event>
<event name='fireAlert'>
  <condition>
    event['SmokeDetector']['smokeDetected'] = 'TRUE'
    and event['SmokeDetector']['temperature'] >= '45'
  </condition>
</event>
```

Figure 6.8: Adaptation trigger events

By using different plug-ins to transform these event definitions into engine-specific CEP rules, we allow the CEVICHE framework to be able to work with virtually any CEP engine, which helps to deal with the **CEP heterogeneity** problem. However there are some functions that may not be covered by the event definition in ABPL, since they are very specific to the engine we want to use. Such is the case of time functions which are not supported by all the engines. In this case we use the

`@system` call, which is taken by the translator *"as is"*, and sent to the CEP engine as an explicit instruction, as is shown in the *radiationAlert* event.

```
insert into radiationAlert(level,location,time)
select a.level, a.location, timeStamp.toString()
from pattern[
  every a = Radiation -> (level >= '20')
];

insert into strongWindAlert(speed,direction)
select a.windSpeed, a.windDirection
from pattern[
  every a = Weather -> (windSpeed >= '25')
];

insert into heavyPrecipitationAlert(rain,snow)
select a.rain, a.snow
from pattern[
  every a = Weather -> (rain >= '25' or snow >= '75')
];

insert into fireAlert()
select 1
from pattern[
  every a = SmokeDetector -> (smokeDetected = 'TRUE' and temperature >= '45')
];
```

Figure 6.9: Scenario events on Esper

To adapt the business process we need to specify the adaptation actions that need to be done. These actions include the addition and removal of activities and/or links that associate different activities. In FIG. 6.10 we show an example of the adaptation tasks executed as a consequence to the *fireAlert* event. One of the advantages of the CEVICHE framework about dynamic adaptation, is that we do not need to add any specific instructions for **undoing the adaptations**, once the triggering event is no longer valid.

For example, in the scenario, when the fire is controlled, the ¬*fireAlert* event is received. This event is identified by CEVICHE as an opposite event, and triggers the adaptation undoing (as previously explained in Chapter 4. The opposite instructions as the ones we defined for the actual adaptation are made, but the subsequent adaptations (in this case the ones related to the *strongWind* event, are reapplied to the process and maintained, since they are still valid).

```
<situation name='explosionRisk'>
  <situation-event>fireAlert</situation-event>
  <process name='CBRN'>
    <adapt>
      <add>
        <activity>evaluateThreat</activity>
        <link from='armyResponse' to='evaluateThreat'/>
        <activity>reportCondition</activity>
        <link from='evaluateThreat' to='reportCondition'/>
        <activity>fightFire</activity>
        <link from='firemenResponse' to='fightFire'/>
        <activity>setNewPerimeter</activity>
        <link from='setNewPerimeter' to='maintainPerimeter'/>
      </add>
      <del>
        <link from='armyResponse' to='armyEvacuate'/>
        <activity>armyEvacuate</activity>
        <activity>armyDistIodine</activity>
        <link from='firemenResponse' to='firemenEvacuate'/>
        <activity>firemenEvacuate</activity>
        <activity>firemenRescue</activity>
      </del>
    <adapt>
  </process>
</situation>
```

Figure 6.10: Response to explosion risk

6.4 Quantitative evaluation

Adding dynamic adaptation to an application, or in this case a business process, brings an additional cost to its execution. However, this cost can be justified by the advantages of being able to manipulate the behavior of the process at run-time, without losing any information and without having any downtime. Moreover, if we compare the time it takes to execute the process against the time it takes to adapt it, the adaptation time becomes negligible.

For evaluating the performance of our solution, we considered three criteria: ***i)*** compare the time it takes for adaptation vs redeploying the business process, ***ii)*** measure the overhead that dynamic adaptation adds to the execution of the business process with respect to the whole time of execution (in different execution environments), and finally, ***iii)*** analyze the cost of undoing adaptations, with respect to the moment in which they are done.

Test Bed Configuration

The CEVICHE Framework consists of 4.5 *KLOC*, and 15 classes developed in Java. The plug-ins have around 0.5 *KLOC* each (for Esper and Etalis), though this may vary from each engine to the other, depending in the complexity of the engine's own language. As for this scenario, it has around 0.5 *KLOC* of BPEL code and 1.2 *KLOC* of WSDL specifications for the Web Services.

The tests were executed using a Mobile Workstation Dell Precision M4400 with an Intel Core 2 Duo Processor T9900 (3 GHz) and 4 GB of RAM, running Ubuntu Linux 11.10. For each of the tests, we show the best case scenario (minimum execution time), the worst case scenario (maximum execution time), and the average of all the execution times. Each test was run 100 times to obtain a valid average result of the execution.

6.4.1 Adaptation VS Redeploy

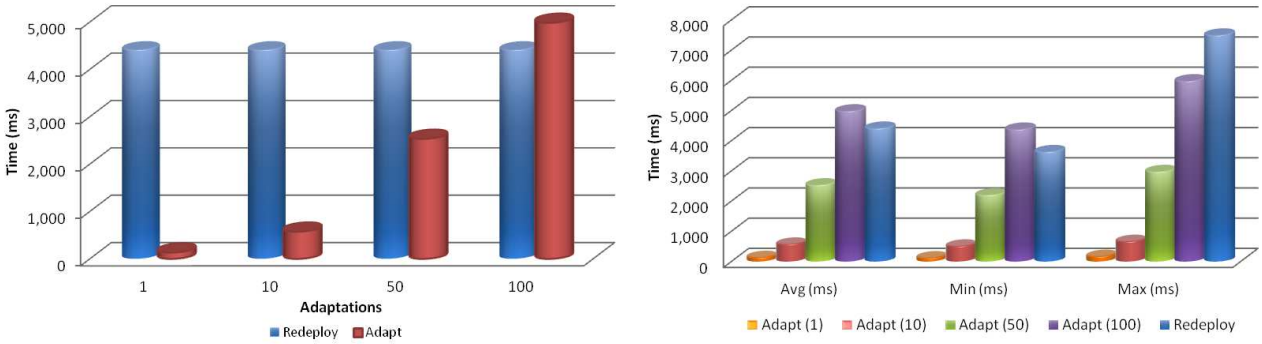
Dynamic adaptation has a cost to the execution of the process. This cost varies depending on the number of changes that we want to do to our business process at the same time. In TAB. 6.2, we show our first evaluation criteria, which is the time it takes for our framework to adapt a process with 1, 10, 50 and 100 changes, and the time it takes to redeploy the whole process. We consider a change as an action that manipulates a business process (*i.e.*, adding or removing activities from it), as presented in TAB. 4.1 (p. 64). To the effect of this test, every change meant substituting one activity by another (adding and activity and removing another).

Time	Adapt (1)	Adapt (10)	Adapt (50)	Adapt (100)	Redeploy
Avg (ms)	131	572	2,532	4,982	4,411
Min (ms)	118	509	2,202	4,376	3,633
Max (ms)	167	663	2,975	5,978	7,495

Table 6.2: Adaptation VS. Redeploy

Using this information, in FIG. 6.11(a) we can see in a graphic form, a comparison between the average time for executing a different number of simultaneous

adaptations, versus the time it takes to redeploy the process. We can notice that adapting is considerably faster than redeploying, with the additional advantage that we do not have any downtime nor do we lose any of the current execution information. In the extreme case of having to do 100 adaptations at the same time (100 changes caused by the same adaptation), according to our measures, then it would be better to redeploy the process, due to the excessive number of changes.



(a) Average time

(b) Max, Min and Avg Measures

Figure 6.11: Adaptation VS. Redeploy

However, it should be noted that if these changes are not made all together, adaptation is still a better option. Moreover, when comparing the minimum, maximum and average execution times of all the adaptation tests against the redeploy time, as shown in FIG. 6.11(b), we can observe that the time to redeploy the process may change considerably from one time to the other. It should be noted that, in the worst-case-scenario, even with 100 simultaneous changes, the longest adaptation time is still faster than the redeploy. In addition to this, it must be considered that the time it takes to manually modify the business process before being able to redeploy it is not being taken into consideration in this comparison, which would favor even more our approach of dynamic adaptation.

6.4.2 Adaptation Overhead

We have already shown that adaptation is most of the times faster and more convenient than a complete process redeployment. For our second criteria we will

show the cost that dynamic adaptation has as an overhead to the business process execution.

For this test, we considered three different execution environments: Local, Network and Internet. In the first one, all the services of the business process are contained locally, and therefore there is no communication outside of the server running the business process. In the second environment, the services are contained in the same local network as the business process server, so the communication is fast and more or less predictable, but still slower than the completely local execution. Finally, the third environment is set by using services contained over the Internet, which brings a big delay and unpredictable execution times. This last environment is the closest one to the reality, where the service providers are usually external entities that are out of the control of the process owner.

On each of these environments we executed different business processes, containing 10, 100 and 500 activities, as is illustrated in TAB. 6.3. The executed processes were automatic and synchronous, which means that there is no delay between the request and the response, other than the time it takes for the server to process the request.

Activities / Time	Local			Network			Internet*		
	10	100	500	10	100	500	10	100	500*
Avg (ms)	430	4,171	19,812	1,590	14,787	70,978	7,360	66,976	364,015
Min (ms)	370	3,293	14,819	1,270	11,049	50,273	5,410	51,395	238,987
Max (ms)	960	10,560	100,320	3,780	41,580	478,170	53,860	297,320	1,722,598

*The Internet test with 500 activities was only run run 50 times due to time constraints.

Table 6.3: Latency of Web Service call

Once we measured the time it takes for the processes to execute, we can calculate the overhead that dynamic adaptation adds to it. For each of the processes, we considered a 10%[26] of adaptations with respect to the total number of activities

[26]These data are merely informative, as the number of simultaneous adaptations may vary widely on each situation, but modifying the 10% of the whole business process at once seemed as a reasonable proportion to create a point of comparison.

in the process (*i.e.*, 10 activities, 1 adapt; 100 activities, 10 adapts; 500 activities, 50 adapts). We present the result of this comparison in TAB. 6.4.

	Local			Network			Internet		
	10	100	500	10	100	500	10	100	500
Adaptation	30.23%	13.71%	12.78%	8.24%	3.87%	3.57%	1.78%	0.85%	0.70%
Execution	69.77%	86.29%	87.22%	91.76%	96.13%	96.43%	98.22%	99.15%	99.30%

Table 6.4: Adaptation overhead

We can note that as the business process becomes more complex, and the communication medium is slower and less reliable, the overhead of adding dynamic adaptation becomes more negligible. As presented in FIG. 6.12, the overhead of adaptation in this case goes from a very considerably 30% in a small local business process, to a negligible less than 1% when running considerably larger processes over the Internet. Nevertheless, these overheads were obtained using automatic and synchronous processes, as we previously stated. In real life, some of the processes are asynchronous, and may take days or even weeks to complete, which makes this overhead practically inexistent.

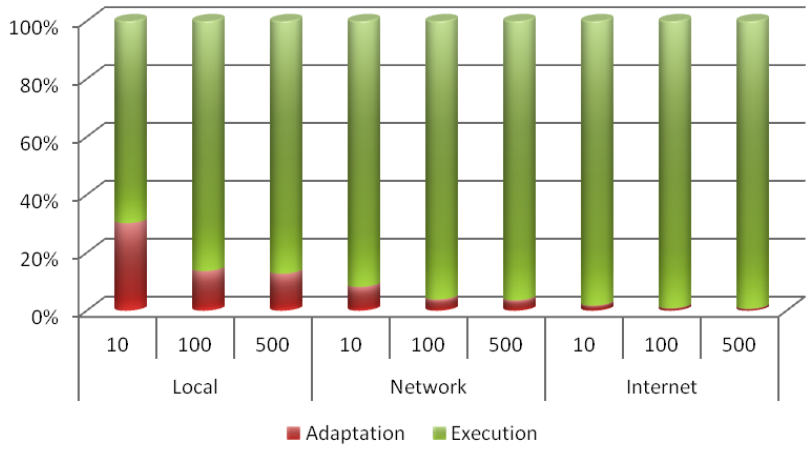


Figure 6.12: Adaptation Time VS. Execution Time

6.4.3 Undoing adaptations

Our final point of evaluation is the cost of undoing adaptations. This task requires to process a considerable amount of information, that depends on the number of changes being made between the moment in which the adaptation happened and the undoing is triggered. This means that undoing an adaptation adds an overhead that can be substantially higher than doing it.

As we presented in Chapter 4, undoing an adaptation with CEVICHE is achieved following four steps: recognize, rewind, prune and replay. The first step (recognize) is done by the CEP engine and works only as the trigger for undoing the adaptation, so it is not considered for measuring the overhead. The other three steps are closely related to the number of changes that have been done to the process before the undoing is triggered, so the time it takes to unadapt depends on the number of adaptations that have to be reconsidered, as can be observed in TAB. 6.5.

Time	After 1 adapt.	After 10 adapt.	After 50 adapt.	After 100 adapt.
Avg (ms)	143	1,350	6,375	12,375
Min (ms)	129	1,121	5,546	10,024
Max (ms)	178	1,620	8,224	17,325

Table 6.5: Cost of undoing adaptations

Even though this overhead is considerably higher than simple adaptations, automatic undoing of adaptations provides the advantage of not having to deal with specific circumstances to manually undo the changes to the process. Moreover, it is still usually less expensive to do/undo adaptations than to redeploy the process every time, and the time it takes to undo an adaptation is still negligible when compared to the whole business process execution time.

6.5 Summary

In this chapter, we have presented a validation for the CEVICHE framework. We described a nuclear crisis management scenario to show how different roles can

interact during a business process and how the behavior of those roles can change during the execution of the process to adapt to new and unforeseen situations. We presented the implementation of some of the adaptation situations, using ABPL to define the events and adaptation actions. We showed how using CEVICHE we help to tackle the main challenges presented throughout this dissertation: **dynamic business process adaptation, context integration, CEP heterogeneity** and **proper undoing of adaptations**.

We evaluated our solution considering the time it takes to dynamically adapt a business process and compared this to the time it takes to redeploy the business process after being modified, and found as a result that dynamic adaptation is the best option, even without considering the time it would take to manually modify an existing process. We measured the overhead that dynamically doing and undoing adaptations add to the execution of a business process, and concluded that as the communication medium is less reliable (as the Internet), and the business process gets more complex, this overhead becomes more negligible, to the point of being less than 1% in an automatic synchronous execution. The overhead is smaller, even insignificant, when dealing with asynchronous business process where the response from the service provider is not automatic and may take several hours or even days.

This chapter concludes the third part of this document, which was dedicated to the validation of our approach. In the following chapter we summarize the main contributions of this dissertation, present the conclusions of the research work, and define a set of perspectives for future work.

Part IV

Conclusion

Chapter 7

Conclusions and Perspectives

"Procrastination is like a credit card: it's a lot of fun until you get the bill."
- Christopher Parker

Contents

7.1 Summary of the Dissertation

Nowadays, business processes have become a crucial part of many organizations, playing every time a more important role in the way they operate. These processes rely on services provided either by their own organizations or by third parties, and are orchestrated using well known standards, such as BPEL, to work together and achieve a common goal. At the same time, the successful execution of these business processes depends on varying amount of factors, not all of which can be previewed beforehand. This translates into a strong need to monitor the environment in which the execution takes place, in order to detect the possible situations that

could affect the performance of the process or even prevent them from reaching its goal.

Nevertheless, identifying these situations is not enough, since there is also a need to respond to those situations in order to continue an optimal execution. Unfortunately, the static nature of business processes prevents them from being able to be modified, without having to be redeployed, thus leading to downtime of the service and loss of information of any current executions. They may, however, include several conditions to make decisions during their execution, considering the current situation in which the process is being executed, but this would lead to more complex processes, that create cross-cutting concerns and hinder maintainability. Moreover, the information obtained in this way will remain scarce, since it will be limited to only certain places of the process and to the specific moment in which it was consulted, as it will not be updated afterwards.

Our research focused on bringing a solution to these problems in the form of the CEVICHE framework. Although there already exist several approaches that intend to bring certain flexibility to the business processes, as we have presented in Section 3, not all of them allow to modify the behavior of the process, and besides they are not well integrated to the context information.

The four main components of the CEVICHE framework (ABPL, Translation Framework, Translation Plug-in and Adaptation Manager), allow it to create context-aware and dynamically adaptable business processes. Our solution benefits from the reconfiguration capabilities provided by SCA, with the FraSCAti platform, as well as from the monitoring capabilities of CEP, without having to depend on specific implementations of the engines, thanks to its plug-in based architecture. This advantage allows for early adoption of the CEP technology without having to deal with the continuous changes and evolution of the domain.

Also, doing and undoing of adaptations is automatically managed by CEVICHE, using the information provided by the user, and keeping a history log of all the adaptations that are being done, so that the user only has to deploy the adaptation rules once and let the framework do the job. This prevents from delayed adaptations due to human interaction and because it is an automatic task, it is also less error-prone. Moreover, we have proven that dynamic adaptation is way better than redeployment of the process in terms of time-cost, and the overhead it brings to normal execution is practically insignificant.

Finally, when comparing our solution to the current State of the Art, we can see that the CEVICHE Framework complies with the description of our intended solution presented in Chapter 3, as can be seen in TAB. 7.1.

Reference	Add / Update / Delete	Context monitoring		Context scope	Automatic adaptation	Proactive adaptation	Adaptation undoing	Adaptation mechanism
CEVICHE	Yes/Yes/Yes	CEP	Events	General	Yes	Yes	Yes	SCA components

Table 7.1: CEVICHE vs. SotA

7.2 Contributions

The contributions of this dissertations can be summarized as follows:

Adaptive Business Process Language (ABPL)

A language that extends the business process definitions with context-awareness and adaptation information (Section 5.4). ABPL works as an add-on to BPEL, where users can express the context information they want to monitor, in the form of events, as well as the adaptation conditions and actions to respond to those changes. It uses data from the business process definition in BPEL, but does not modify the original information, which helps maintainability, as well as a better separation of concerns, since we can leave only the core business logic in the BPEL process, without including all the specific situations.

Plug-in approach for CEP adoption

Our plug-in approach facilitates the integration of CEP with existing business processes and makes the use of this technology easier, without the drawbacks of early adoption and without dealing with CEP heterogeneity (Section 5.6). A plug-in uses the information provided by the user in the ABPL and translates it to an engine-specific language, which prevents the user from having to rewrite all the rules related to finding an adaptation situation.

The advantage of this approach is that a plug-in only needs to be created once for each existing CEP engine, and once it has been developed, it can be shared by all the people using the same engine. And even when people do not use the same engine, they could share their event definitions in ABPL with others (*e.g.,* business partners). Another advantage of this approach is that, if at any moment a standard is defined by the *Event Processing Technical Society*, there only has to be one more plug-in developed for the new standard and we could still be able to use our previously defined rules with any standard compliant engine, without the need to re-write them.

Dynamic adaptation management

The Adaptation Manager built in the CEVICHE framework is probably the must important contributions of this dissertation (Chapter 4). It is based on the reconfiguration capabilities provided by FraSCAti, which allows CEVICHE to be able to dynamically adapt a business process that has previously been transformed into SCA components. The Adaptation Manager is in charge of dealing with all the adaptation information and the logic for doing and undoing changes to the business process. It receives all the adaptation information from the rules provided by the user using ABPL, and stores them in its *rule repository*, where it is consulted for adaptation verifications.

Not only does it executes all the adaptation tasks (*i.e.,* adding or removing activities or links), but it also manages the undoing of those adaptations in order to guarantee a clean and correct unadaptation, when the adapting condition is no longer valid. To achieve this, it follows four mechanisms: *identify*, *rewind*, *prune* and *replay*. Our approach not only considers the original adaptation, but also the subsequent adaptations that were related to it and evaluates its relevance after the unadaptation. In CEVICHE, both doing and undoing adaptations are fully automated, which set the user free from having to deal with any adaptation logic during the execution of the process.

7.3 Perspectives

Although the work presented in this dissertation covers the needs of adding flexibility and context-awareness to business process, there is still some work that could

be done to improve our research. In this section we present some short–term and long–term perspectives that should be considered in the continuation of this work.

Short-term perspectives

Implementation of more plug-ins

There are currently only two plug-ins implemented for the CEP engines (Esper and Etalis). Both are open source, which facilitates the access to their API and allows for faster development. Etalis is a minor alternative, mainly used in research. On the other hand, Esper is widely used in research and industry, however it still remains a small player in the overall marketshare when compared to the alternatives from the big companies. The implementation of plug-ins for the bigger engines developed by the industry (*e.g.,* IBM, Microsoft, Oracle, TIBCO, etc.) is important for incrementing the utility of this approach.

Add family-specific tags to ABPL

As we discussed in Section 2.3, there are three main families of CEP languages: stream-oriented, rule-oriented and imperative. With languages of the same family there is usually a strong resemblance, but they are different from languages from other families. In ABPL we support *system* calls, which is not interpreted by the plug-in, and sent directly to the engine *"as is"*. However, an interesting improvement would be to consider special tags for a certain family, in which case all the languages that support the same type of *system* calls (*e.g.,* system time, date, etc.) could benefit from these tags.

Graphical User Interface

The ABPL is an XML-based language, which is sometimes not easy to manage and edit by hand. However, as XML-based it benefits from an easiness to be parsed and understood by a machine, which could allow a Graphical User Interface (GUI) to be easily implemented and that would simplify the task of creating event and adaptation rules to the user. It could take the information of the BPEL process as input to give options to the user about the existing and alternative activities.

Support fine-grained adaptation

When adapting the business process, in CEVICHE we either add or delete activities and links from the business process. When we want to make simple changes to an activity (*e.g.,* change the service provider), it has to be replaced by another one with the new behavior. In several cases this may be an *overkill*, which leads to more time-consuming adaptations. There should be a way to allow more fine-grained adaptation, which will allow only to change certain properties of an activity without the need to replace it completely. The capabilities to achieve this are already provided by the FraSCAti platform, however the ABPL part dealing with adaptations should be extended to provide this functionality.

Long-term perspectives

Add a monitor for the current state of the process

As we have mentioned throughout the dissertation, the management of adaptations is completely automated in CEVICHE. However, one drawback about this is that there is until now no way to know the current state of an adapted business process, other than the history of changes that have been applied to it. This means that if we want to know how the current process looks like, we would have to follow manually each of the changes that have been done to it, in order to see how it would behave. This is a problem for debugging special and unforeseen situations, since we have to follow each step to know why it is reacting the way it is. There should be a monitor which indicates the current state of the process in a way that can be understood by any user or developer (maybe even a graphical representation of the business process).

Consider instance state while adapting

While doing and undoing adaptations of the business process, CEVICHE considers only the main process definition, and does not consider the state of the current instances of the process. Some considerations are actually done to avoid having an erroneous state in the instance executing the process, *e.g.,* when deleting activities, the activities are not really deleted, but all entry links to them are removed and linked to its subsequent activity. This prevents the system to delete an activity that

is currently being used. However, since each process instance has a context of its own, adapting or unadapting cannot always be performed. To correctly achieve (un)adaptation at the instance level, we need to consider also the step of the process that the instance is running and then adapt its process only when referring to future steps. Also, a particular attention must be paid when loops are involved in the instance, since in this case a previous step is also a future step, and these adaptations need to be managed carefully.

A DSPL approach for adaptations

A Software Product Line (SPL) is a set of systems that share a group of manageable features, where a feature is a characteristic of a system [Pohl et al., 2005]. Dynamic Software Product Lines (DSPL) focus on the development of software products that can be adapted at run-time depending on the requirements of the users and the conditions of the environment [Hallsteinsen et al., 2008]. In CEVICHE, the adaptations are done automatically, however the instructions of how to adapt the process at a given time are provided by the user, and the framework does not reason on whether an adaptation could lead to an undesired result of the process. Using a DSPL approach, we could limit the possible adaptations that can be done to the process in a more business-oriented way. We could, for example, mark activities as optional or mandatory, and in this way prevent an activity from being removed when it is crucial to the success of the process, or we could also prevent that an activity is followed by another when they are not compatible.

Appendix: French Summary

Appendix A

Introduction

Afin de maintenir un niveau compétitif, les organisations utilisent de plus en plus des solutions orientées services pour automatiser et faciliter l'intégration de leurs processus métiers. Ces solutions reposent principalement sur des normes bien connues, telles que BPEL[27] (*Business Process Execution Language*), pour orchestrer les services lors de l'exécution du processus. Dès que les processus métiers évoluent et deviennent plus complexes, les données à traiter augmentent de façon exponentielle, et des facteurs de plus en plus nombreux peuvent détériorer la bonne exécution du processus.

Les informations externes à l'exécution des processus métiers sont connues sous le nom d'informations liées au contexte [Dey et al., 2001]. Étant donné que ces informations sont en constante évolution, il est important d'être capable de les surveiller, afin d'identifier quand des situations particulières se développent, ce qui pourrait affecter l'exécution du processus. Malheureusement, la spécification BPEL ne fournit pas de mécanismes standard permettant de surveiller les informations du contexte. Bien que cela puisse être fait en ajoutant des activités de surveillance spécifiques tout au long du processus, cela ne ferait que conduire à une solution étroite qui permettrait la gestion d'informations spécifiques du contexte liées à certaines parties du processus, en plus de créer plus de problèmes propres à cette solution (par exemple, la maintenabilité, les préoccupations transversales, etc).

[27] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

Néanmoins, il existent des techniques qui devraient permettre la surveillance de ces informations sans interférer avec l'exécution du processus. Une de ces techniques est le "Traitement des Événements Complexes", ou *Complex Event Processing* (CEP) en anglais. Le CEP est une technologie émergente qui peut aider les organisations à bénéficier des informations du contexte, car elle leur permet de trouver en temps réel les relations entre les différents événements, en utilisant des éléments tels que le temps, la causalité, et l'appartenance dans un flux de données pour extraire des informations pertinentes [Luckham, 2002]. Le CEP est utilisé dans une grande variété d'applications, comme la prévention du vol de marchandises [Huber and Michael, 2007], la surveillance du marché boursier [Mangkorntong, 2009], et l'interaction avec les systèmes RFID [Zang et al., 2008].

Cependant, le suivi des informations du contexte pour identifier les situations particulières ne suffit pas. Nous avons de plus besoin d'être en mesure de répondre à ces situations en temps réel. Les définitions de processus métiers sont statiques par nature, ce qui signifie qu'ils ne peuvent pas être modifiés lors de l'exécution. En même temps, il y a un fort besoin d'être en mesure d'adapter ces processus dynamiquement, afin de répondre aux conditions changeantes.

Les modifications manuelles des processus prennent beaucoup de temps, et elles sont de plus sujettes aux erreurs. Par ailleurs, le redéploiement d'une modification du processus métier conduirait à un temps d'indisponibilité du service et à la perte des informations de toutes les instances en cours d'exécution du processus. Dans cette thèse, nous explorons les possibilités de la reconfiguration dynamique, fournies par l'"Architecture des Services à base de Composants", ou *Service Component Architecture* (SCA) en anglais, comme une solution à ce problème [Beisiegel et al., 2007]. Notre proposition vise à intégrer les avantages du CEP pour contrôler les informations du contexte, ainsi que les capacités de reconfiguration de SCA dans les processus métiers existants.

Structure du chapitre

Le reste de ce chapitre introductif est organisé comme suit: Dans la Section A.1, nous identifions les problèmes qui motivent ce travail de recherche. Ensuite, dans la Section A.2, nous présentons nos objectifs de recherche. La Section A.3 explique les contributions apportées par cette thèse. Dans la Section A.4 nous donnons une

brève introduction à chacun des chapitres du document. Finalement, dans la Section A.5, nous énumérons les publications faites au cours du développement de ce travail.

A.1 Compréhension du probllème

Comme nous l'avons expliqué précédemment, il existe actuellement de nombreux problèmes qui entravent la bonne exécution des processus métiers, spécialement lorsque l'on considère la grande dynamicité des environnements dans lesquels ils doivent être exécutés. Au cours de cette thèse, nous avons identifié les questions suivantes:

Difficulté à intégrer les informations du contexte dans les processus métiers

Les processus métiers n'ont pas les capacités nécessaires pour surveiller en permanence les informations autour d'eux. Ils ont une portée limitée sur l'information qu'ils peuvent obtenir et le moment où ils y ont accès. Aujourd'hui, grâce à la croissance des environnements pervasifs et ubiquitaires, nous avons accès à un nombre de plus en plus grand de sources d'information qui pourraient être utiles à l'exécution des processus, mais à partir auxquelles ils ne peuvent bénéficier en raison des contraintes de la spécification de BPEL.

La nature statique des processus métiers

En plus de leur manque de capacités de surveillance, les processus métiesr sont statiques par nature, ce qui signifie qu'ils ne peuvent pas être modifiés lors de l'exécution. Ils n'étaient pas destinés à être modifiés dynamiquement, car ils ont été pensés simplement comme une séquence prédéfinie d'activités visant à atteindre un objectif. Il y a effectivement une certaine flexibilité en termes de modification des prestataires de services, si elles sont spécifiées à l'avance dans la définition des processus métiers, mais nous ne pouvons pas changer le comportement du processus en soi (ajouter de nouvelles activités ou supprimer celles qui existent déjà).

L'adoption d'une nouvelle technologie

Nous avons déjà mentionné l'utilisation du CEP comme une solution à la surveillance des informations du contexte. Cependant, cette technologie est dans un état précoce, et comme telle, elle est encore sujette à évolution. Le CEP a attiré l'attention de plusieurs développeurs de différents domaines, tant dans la recherche comme dans l'industrie, ce qui a conduit à la création de nombreuses implémentations différentes. Mais il n'existe pas encore une norme qui détermine la façon correcte de définir les règles pour le moteur CEP, et chaque mise en œuvre utilise son propre langage. Alors, comment pouvons-nous profiter des avantages du CEP, sans devoir traiter avec les inconvénients d'une technologie encore immature et en pleine évolution?

Défaire une adaptation pourrait être dangereux

Le dernier problème n'est pas spécifique aux processus métiers, mais en fait un problème commun à toutes les solutions d'adaptations dynamiques. Une fois que nous sommes en mesure de gérer l'adaptation dynamique, nous avons également besoin de défaire plus tard cette adaptation parce que les conditions d'adaptation ne sont plus valables. Cela est généralement considéré comme une tâche facile, cependant, si elle ne se fait pas de manière appropriée, elle peut conduire à des états contradictoires des applications, ou dans notre cas, des processus métiers.

A.2 Objectifs de cette thèse

Étant donné les problèmes présentés dans la section précédente, les objectifs de cette thèse sont destinés à leur apporter une solution. Par notre proposition, nous prévoyons d'améliorer l'exécution des processus métiers en leur offrant une sensibilité au contexte, l'adaptation dynamique, et la possibilité de défaire correctement et automatiquement les adaptations. En même temps, nous voulons fournir une solution indépendante de la plateforme de traitement d'événements qui donne à l'utilisateur la possibilité de choisir le meilleur moteur CEP à sa convenance, sans avoir besoin de se soucier des différents langages ou de leurs implémentations.

Sensibilisation au contexte

Afin d'améliorer la surdité relative aux informations du contexte qui existe actuellement dans les processus métiers, nous allons intégrer le CEP comme une source externe d'information, qui aura la tâche de surveiller constamment les changements de l'environnement et de détecter le moment où une adaptation est nécessaire dans le processus pour arriver à une exécution optimale. En tant que source externe, cela n'affectera ni la performance ni la maintenabilité du processus original, et fournira un moyen simple pour mettre à jour les situations que nous voulons surveiller.

L'adaptation dynamique

Une partie fondamentale de notre proposition est la dynamicité, et par dynamicité nous entendons le fait d'être capable d'adapter les processus métiers automatiquement au moment de l'ex'ecution, sans aucune interruption ni perte d'information. Nous devons être en mesure de répondre rapidement aux changements du contexte afin de garantir que notre processus est exécuté dans les meilleures conditions possibles pour atteindre son objectif.

Indépendance de la plateforme

Le choix est un atout très précieux. La possibilité de pouvoir changer d'une option à l'autre, surtout quand le marché de ces options est en changement continu, est extrêmement importante. Dans le cas du CEP, le changement d'une solution à une autre impliquerait effectivement refaire tous les règles définies pour la surveillance dans un nouveau langage, qui a parfois une syntaxe complètement différente.

Défaire correctement et automatiquement les adaptations

Lorsque nous abordons la sensibilité au contexte et l'adaptation dynamique, nous devons également tenir compte de la capacité à annuler les modifications faites sur le processus métier lorsque les conditions d'adaptation ne sont plus valables. Ce n'est pas une tâche simple, ni triviale, car toutes les adaptations ultérieures et dépendantes doivent également être prises en compte lors de l'annulation d'une adaptation.

A.3 Contribution

Afin d'améliorer la compréhension de notre travail, dans cette section, nous décrivons brièvement les principales contributions de cette thèse. Comme indiqué précédemment, l'objectif de notre travail est de fournir la capacité aux processus métiers de devenir sensibles au contexte et dynamiquement adaptables. Pour y parvenir, nous avons créé le cadre logiciel CEVICHE. Notre première contribution est un langage adaptatif, appelé *"Adaptive Business Process Language"* (ABPL), qui permet à l'utilisateur de définir les besoins d'adaptation, en répondant aux quatre questions d'adaptation: *Qu'*est-ce qu'il faut adapter?, *Quand* faut-il adapter?, *Où* adapter?, et *Comment* adapter? Le langage ABPL fusionne les définitions d'adaptation et celles des événements pour simplifier son utilisation sur un processus métier.

La deuxième contribution de cette thèse est de fournir une approche à base de plug-ins qui permettra au cadre logiciel CEVICHE d'interagir avec n'importe quel moteur de CEP existant pour l'intégration du contexte. Un plug-in utilise les informations fournies par l'utilisateur dans le langage ABPL et les traduit dans le langage spécifique du moteur, ce qui empêche l'utilisateur d'avoir à réécrire toutes les règles relatives à la recherche d'une situation d'adaptation.

La troisième contribution, et probablement la plus importante de notre travail, est un Gestionnaire d'Adaptation, appelé *Adaptation Manager* en anglais, construit dans le cadre logiciel CEVICHE. Il gère la manipulation effective du processus métier et considère les informations provenant des moniteurs de contexte pour déclencher les adaptations. Il stocke les informations d'adaptation afin d'être en mesure de déterminer quand une condition d'adaptation n'est plus valide et déclencher l'annulation de l'adaptation. Finalement, il gère l'annulation des adaptations en tenant compte des adaptations précédentes causalement liées et qui doivent être examinées.

A.4 Organisation du document

Cette thèse est divisée en quatre parties. La première partie, l'Etat de l'Art, donne les bases du domaine dans lequel notre travail a lieu, et analyse certains travaux connexes. La deuxième partie, intitulée Contribution, présente notre travail de

manière plus détaillée. Dans la troisième partie, intitulée Validation, nous présentons certaines mesures et la mise en œuvre d'un scénario de cas d'utilisation. Dans la partie finale, la Conclusion, nous résumons nos travaux et discutons les perspectives.

Partie I: Etat de l'Art

Chapitre 2: Contexte et concepts. Dans ce chapitre, nous donnons une brève introduction à un ensemble de domaines utilisés tout au long de la thèse, afin de permettre une meilleure compréhension du contexte dans lequel notre travail a eu lieu. Nous définissons la terminologie et les concepts présentés dans les chapitres suivants.

Chapitre 3: L'adaptation et les processus métiers. Dans ce chapitre, nous présentons plusieurs travaux connexes qui ont essayé de résoudre le manque de flexibilité dans les processus métiers avec des approches différentes. Nous comparons ces approches en utilisant différents critères liés au type d'adaptation qu'ils utilisent et les mécanismes pour y parvenir.

Partie II: Contribution

Chapitre 4: Processus métiers dynamiquement adaptables a base des événements. Dans ce chapitre, nous présentons notre solution pour adapter dynamiquement les processus métiers en utilisant une approche á base d'événements pour fournir les informations de contexte. Nous montrons aussi que défaire ces adaptations n'est pas une tâche triviale et de nous présentons notre proposition pour défaire correctement une adaptation basée sur des événements, d'une manière propre et automatique.

Chapitre 5: Le cadre logiciel CEVICHE. Dans ce chapitre, nous présentons notre mise en œuvre pour faire et défaire des adaptations dynamiques, appelée le Cadre logiciel CEVICHE. Nous utilisons une approche basée sur des composants pour fournir de la dynamicité aux processus métiers. Nous utilisons aussi le traitement des événements complexes (CEP) comme un moyen de faire face à des informations du contexte. Enfin, nous présentons également notre langage d'adaptation, l'ABPL, et nous montrons un exemple de l'utilisation des plug-ins.

Partie III: Validation

Chapitre 6: Validation. Dans ce chapitre, nous validons notre travail en utilisant un scénario de gestion de crise nucléaire, et nous présentons les résultats de nos tests pour démontrer pourquoi les adaptations dynamiques sont la meilleure solution et comment leur coût peut être négligeable.

Partie IV: Conclusion

Chapitre 7: Conclusions et perspectives. Dans ce chapitre, nous présentons les conclusions de nos travaux et nous présentons quelques perspectives à court et à long terme.

A.5 Publications

Ci-dessous nous vous présentons une liste des publications de recherche issues de notre travail autour de cette thèse.

Revues internationales

- Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy and Frank Eliassen. *The DigiHome Service-Oriented Platform*. Software: Practice and Experience (SP&E). 2012. Pages 17. *À apparaître*.
  Classement (CORE): A

- Gabriel Hermosillo, Sébastien Mosser, Lionel Seinturier, Laurence Duchien. CEVICHE*: A Framework for Dynamically Doing and Undoing Adaptations in Context-Aware Business Process with an Event-Driven Approach*. Journal of Systems and Software (JSS). 2012. Pages 25. *Soumis*.
  Classement (CORE): A

Conférences internationales

- Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle, Lionel Seinturier. *A Middleware Platform to Federate Complex Event Processing*. The Sixteenth IEEE International EDOC Conference (EDOC'12). Pages 10. Beijing, Chine. Septembre 2012. *À apparaître*.
  Classement: (CORE): B

- Sébastien Mosser, Gabriel Hermosillo, Anne-Françoise Le Meur, Lionel Seinturier, Laurence Duchien. *Undoing Event-Driven Adaptation of Business Processes*. The 8th IEEE 2010 International Conference on Services Computing (SCC'11). Pages 234–241. Washington D.C., USA. Juillet 2011.
  Taux d'acceptation: 17%, Classement (CORE): A

- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Creating Context-Adaptive Business Processes*. The 8th International Conference on Service Oriented Computing (ICSOC'10). Pages 228–242. San Francisco, Calif., USA. Décembre 2010.
  Taux d'acceptation: 15%, Classement (CORE): A

- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Using Complex Event Processing for Dynamic Business Process Adaptation*. The 7th IEEE International Conference on Services Computing (SCC'10). Pages 466–473. Miami, Florida, USA. Juillet 2010.
  Taux d'acceptation: 18%, Classement (CORE): A

- Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy and Frank Eliassen. *RESTful Integration of Heterogeneous Devices in Pervasive Environments*. 10th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'10). Pages 1–14. Amsterdam, Pays-Bas. Juin 2010.
  Taux d'acceptation: 32%, Classement (CORE): B

- Patricia Jaimes, Gabriel Hermosillo, Roberto Gómez. *Una marca de agua inteligente aplicada al dinero electrónico*. The Fifth Ibero-American Congress on Information Security (CIBSI'09). Pages 225–239. Montevideo, Uruguay. Novembre 2009.

Ateliers internationaux

- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Complex Event Processing for Context-Adaptive Business Processes*. The 8th BElgian-NEtherlands software eVOLution seminar (BENEVOL'09). Louvain-la-neuve, Belgique. Décembre 2009.
- Gabriel Hermosillo, Julien Ellart, Lionel Seinturier, Laurence Duchien. *A Traceability Service to Facilitate RFID Adoption in the Retail Supply Chain*. The 3rd International Workshop on RFID Technology - Concepts, Applications, Challenges (IWRT'09). Pages 49–58. Milan, Italie. Mai 2009.

Affiches

- Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle, Lionel Seinturier. *Distributed Complex Event Processing Engine*. Génie de la Programmation et du Logiciel (GPL'12). Rennes, France. Juin 2012.
- Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien. *Using* CEP *to create context-adaptive processes in pervasive environments*. CANOE and EuroSys Summer School. Oslo, Norvège. Août 2009.

Bibliography

[Adi and Etzion, 2004] Adi, A. and Etzion, O. (2004). Amit - the situation manager. *The VLDB Journal*, 13:177–203. 60

[Baligand et al., 2007] Baligand, F., Rivierre, N., and Ledoux, T. (2007). A declarative approach for qos-aware web service compositions. In *Proceedings of the 5th international conference on Service-Oriented Computing*, ICSOC '07, pages 422–428, Berlin, Heidelberg. Springer-Verlag. 22

[Bastida et al., 2008] Bastida, L., Nieto, F. J., and Tola, R. (2008). Context-aware service composition: a methodology and a case study. In *Proceedings of the 2nd international workshop on Systems development in SOA environments*, SDSOA '08, pages 19–24, New York, NY, USA. ACM. 38, 41

[Beisiegel et al., 2007] Beisiegel, M., Booz, D., Colyer, A., Hildebrand, H., Marino, J., and Tam, K. (2007). Sca service component architecture. http://www.osoa.org/. 2, 25, 87, 140

[Bernal et al., 2010] Bernal, J. F. M., Falcarin, P., Morisio, M., and Dai, J. (2010). Dynamic Context-aware Business Process: a Rule-based Approach Supported by Pattern Identification. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC)*, pages 470–474. ACM. 51

[Bernstein et al., 1987] Bernstein, P. A., Hadzilacos, V., and Goodman, N. (1987). *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 71

[Blanc et al., 2008] Blanc, X., Mounier, I., Mougenot, A., and Mens, T. (2008). Detecting Model Inconsistency through Operation-Based Model Construction. In *30th International Conference on Software Engineering (ICSE 2008)*, pages 511–520. ACM. 63

[Canfora et al., 2008] Canfora, G., Di Penta, M., Esposito, R., and Villani, M. L. (2008). A framework for QoS-aware binding and re-binding of composite web services. *Journal of Systems and Software*, 81:1754–1769. 38

[Chandy and Schulte, 2010] Chandy, K. M. and Schulte, W. R. (2010). *Event Processing - Designing IT Systems for Agile Companies*. McGraw-Hill. 32, 33, 64

[Chandy et al., 2011] Chandy, M. K., Etzion, O., and von Ammon, R. (2011). 10201 Executive Summary and Manifesto – Event Processing. In Chandy, K. M., Etzion, O., and von Ammon, R., editors, *Event Processing*, number 10201 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. 66

[Chappell, 2007] Chappell, D. (2007). Introducing SCA. http://www.davidchappell.com/articles/introducing_sca.pdf. 25

[Charfi et al., 2009] Charfi, A., Dinkelaker, T., and Mezini, M. (2009). A Plug-in Architecture for Self-Adaptive Web Service Compositions. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 35–42. IEEE Computer Society. 44, 47

[Charfi and Mezini, 2007] Charfi, A. and Mezini, M. (2007). AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web*, 10(3):309–344. 47

[Cheng et al., 2009] Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., and Magee, J., editors (2009). *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*. Springer. 36

[Christos et al., 2009] Christos, K., Vassilakis, C., Rouvas, E., and Georgiadis, P. (2009). Qos-driven adaptation of bpel scenario execution. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 271–278, Washington, DC, USA. IEEE Computer Society. 22

[Colombo et al., 2006] Colombo, M., Di Nitto, E., and Mauri, M. (2006). SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *Service-Oriented Computing - ICSOC 2006*, volume

4294 of *Lecture Notes in Computer Science*, pages 191–202. Springer Berlin / Heidelberg. 39, 41

[Courbis and Finkelstein, 2005] Courbis, C. and Finkelstein, A. (2005). Towards aspect weaving applications. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 69–77, New York, NY, USA. ACM. 36, 62

[Cruz Torres et al., 2010] Cruz Torres, M. H., Noël, V., Holvoet, T., and Arcangeli, J.-P. (2010). MAS organisations to adapt your composite service. In *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, MONA '10, pages 33–39, New York, NY, USA. ACM. 39

[da Silva et al., 2010] da Silva, M. A. A., Mougenot, A., Blanc, X., and Bendraou, R. (2010). Towards Automated Inconsistency Handling in Design Models. In *Advanced Information Systems Engineering, 22nd International Conference (CAiSE 2010)*, volume 6051 of *Lecture Notes in Computer Science*, pages 348–362. Springer. 50, 51

[Dey et al., 2001] Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:97–166. 2, 28, 139

[Douence et al., 2001] Douence, R., Motelet, O., and Südholt, M. (2001). A Formal Definition of Crosscuts. In *Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, REFLECTION '01, pages 170–186. Springer-Verlag. 36

[Edwards, 2007] Edwards, M. (2007). Relationship between SCA and BPEL. Technical report, Open Service Oriented Architecture Collaboration (OSOA). 87

[Erl, 2007] Erl, T. (2007). *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA. 15

[Etzion and Niblett, 2010] Etzion, O. and Niblett, P. (2010). *Event Processing in Action*. Manning Publications Co. 28, 31, 65

[Fielding, 2000] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine. 17

[Geebelen et al., 2010] Geebelen, K., Kulikowski, E., Truyen, E., and Joosen, W. (2010). A MVC Framework for Policy-Based Adaptation of Workflow Processes: A Case Study on Confidentiality. In *Proceedings of the 2010 IEEE International Conference on Web Services*, ICWS '10, pages 401–408. 46, 49

[Hallsteinsen et al., 2008] Hallsteinsen, S., Hinchey, M., Park, S., and Schmid, K. (2008). Dynamic Software Product Lines. *Computer*, 41(4):93–95. 135

[Hermosillo et al., 2010a] Hermosillo, G., Seinturier, L., and Duchien, L. (2010a). Creating Context-Adaptive Business Processes. In *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC 2010)*, volume 6470 of *Lecture Notes in Computer Science*, pages 228–242. Springer Berlin / Heidelberg. 84

[Hermosillo et al., 2010b] Hermosillo, G., Seinturier, L., and Duchien, L. (2010b). Using Complex Event Processing for Dynamic Business Process Adaptation. In *Proceedings of the 2010 IEEE International Conference on Services Computing*, SCC '10, pages 466–473, Washington, DC, USA. IEEE Computer Society. 84

[Huber and Michael, 2007] Huber, N. and Michael, K. (2007). Minimizing Product Shrinkage across the Supply Chain using Radio Frequency Identification: a Case Study on a Major Australian Retailer. In *ICMB '07: Proceedings of the International Conference on the Management of Mobile Business*, page 45. IEEE Computer Society. 2, 140

[Janiesch et al., 2011] Janiesch, C., Matzner, M., and Müller, O. (2011). A Blueprint for Event-Driven Business Activity Management. In *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 17–28. Springer Berlin / Heidelberg. 53

[Josuttis, 2007] Josuttis, N. (2007). *Soa in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc. 15, 16

[Juhnke et al., 2010] Juhnke, E., Do Andrnemann, T., Kirch, S., Seiler, D., and Freisleben, B. (2010). Simplebpel: Simplified modeling of bpel workflows for scientific end users. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pages 137 –140. 22

[Klein et al., 2009] Klein, J., Kienzle, J., Morin, B., and Jézéquel, J.-M. (2009). Aspect Model Unweaving. In *Model Driven Engineering Languages and Systems, 12th In-*

ternational Conference (MODELS 2009), volume 5795 of *Lecture Notes in Computer Science*, pages 514–530. Springer. 51, 78

[Koning et al., 2009] Koning, M., Sun, C.-a., Sinnema, M., and Avgeriou, P. (2009). VxBPEL: Supporting variability for Web services in BPEL. *Information and Software Technology*, 51(2):258–269. 44

[Kramer, 2008] Kramer, B. (2008). Component meets service: what does the mongrel look like? *Innovations in Systems and Software Engineering*, 4:385–394. 87

[Léger et al., 2010] Léger, M., Ledoux, T., and Coupaye, T. (2010). Reliable Dynamic Reconfigurations in a Reflective Component Model. In *Component-Based Software Engineering, 13th International Symposium (CBSE 2010)*, volume 6092 of *Lecture Notes in Computer Science*, pages 74–92. Springer. 50

[Leitner et al., 2010] Leitner, P., Michlmayr, A., Rosenberg, F., and Dustdar, S. (2010). Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In *Proceedings of the 2010 IEEE International Conference on Web Services*, ICWS '10, pages 369–376, Washington, DC, USA. IEEE Computer Society. 45

[Lin et al., 2008] Lin, H., Liu, S., and Fan, Y. (2008). Service-oriented enterprise cooperation: Modeling method and system. In *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pages 1032 –1037. 22

[Lins et al., 2007] Lins, F. A. A., dos Santos Júnior, J. C., and Rosa, N. S. (2007). Adaptive web service composition. *ACM SIGSOFT Software Engineering Notes*, 32. 39

[Luckham, 2002] Luckham, D. C. (2002). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc. 2, 30, 66, 86, 140

[Majernik et al., 2011] Majernik, F., Jensen, M., and Schwenk, J. (2011). Marv - data level confidentiality protection in bpel-based web service compositions. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1 –8. 22

[Mangkorntong, 2009] Mangkorntong, P. (2009). *A Domain-Driven Approach for Detecting Event Patterns in E-Markets: A Case Study in Financial Market Surveillance*. VDM Verlag, Saarbrücken, Germany, Germany. 2, 140

[Marconi et al., 2009] Marconi, A., Pistore, M., Sirbu, A., Eberle, H., Leymann, F., and Unger, T. (2009). Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation. In *Service-Oriented Computing*, volume 5900 of *Lecture Notes in Computer Science*, pages 445–454. Springer Berlin / Heidelberg. 45

[McKinley et al., 2004a] McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. C. (2004a). A Taxonomy of Compositional Adaptation. Technical report, Michigan State University. 92, 95

[McKinley et al., 2004b] McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. C. (2004b). Composing Adaptive Software. *Computer*, 37:56–64. 92, 95

[Mosser et al., 2011] Mosser, S., Hermosillo, G., Le Meur, A.-F., Seinturier, L., and Duchien, L. (2011). Undoing event-driven adaptation of business processes. In *Proceedings of the 2011 IEEE International Conference on Services Computing*, SCC '11, pages 234–241, Washington, DC, USA. IEEE Computer Society. 71

[Mukherjee et al., 2008] Mukherjee, D., Jalote, P., and Gowri Nanda, M. (2008). Determining qos of ws-bpel compositions. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, pages 378–393, Berlin, Heidelberg. Springer-Verlag. 22

[Nassar et al., 2009] Nassar, P. B., Badr, Y., Biennier, F., and Barbar, K. (2009). Extended bpel with heterogeneous authentication mechanisms in service ecosystems. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '09, pages 19:126–19:133, New York, NY, USA. ACM. 22

[OASIS, 2006] OASIS (2006). Reference Model for Service Oriented Architecture 1.0. Website http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf. 15

[OASIS, 2007] OASIS (2007). OASIS Standard. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html. 20

[O'Keefe, 1990] O'Keefe, R. A. (1990). *The craft of Prolog*. MIT Press, Cambridge, MA, USA. 64

[Papazoglou, 2008] Papazoglou, M. P. (2008). *Web Services: Principles and Technology*. Pearson, Prentice Hall. 36

[Paschke and Kozlenkov, 2009] Paschke, A. and Kozlenkov, A. (2009). Rule-Based Event Processing and Reaction Rules. In *Proceedings of the 2009 International Symposium on Rule Interchange and Applications*, RuleML '09, pages 53–66, Berlin, Heidelberg. Springer-Verlag. 32

[Peltz, 2003] Peltz, C. (2003). Web Services Orchestration and Choreography. *Computer*, 36:46–52. 19

[Pohl et al., 2005] Pohl, K., Böckle, G., and Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag. 38, 135

[Qiu et al., 2007] Qiu, Z., Zhao, X., Cai, C., and Yang, H. (2007). Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 973–982, New York, NY, USA. ACM. 20

[Rahman et al., 2008] Rahman, S. S. u., Aoumeur, N., and Saake, G. (2008). An adaptive ECA-centric architecture for agile service-based business processes with compliant aspectual .NET environment. In *iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 240–247. ACM. 39, 41

[Rouvoy et al., 2009] Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S. O., Lorenzo, J., Mamelli, A., and Scholz, U. (2009). MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. *Software Engineering for Self-Adaptive Systems*, pages 164–182. 36

[Sánchez and Villalobos, 2008] Sánchez, M. and Villalobos, J. (2008). A flexible architecture to build workflows using aspect-oriented concepts. In *AOM '08: Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling*, pages 25–30. ACM. 44

[Schilit and Theimer, 1994] Schilit, B. N. and Theimer, M. M. (1994). Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32. 28

[Seinturier et al., 2009] Seinturier, L., Merle, P., Fournier, D., Dolet, N., Schiavoni, V., and Stefani, J.-B. (2009). Reconfigurable SCA Applications with the FraSCAti Platform. In *Proceedings of the 2009 IEEE International Conference on Services Computing*, SCC '09, pages 268–275. 27, 88

[Seinturier et al., 2012] Seinturier, L., Merle, P., Rouvoy, R., Romero, D., Schiavoni, V., and Stefani, J.-B. (2012). A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. *Software: Practice and Experience*. 27, 88

[Sharon and Etzion, 2008] Sharon, G. and Etzion, O. (2008). Event-processing Network Model and Implementation. *IBM Systems Journal*, 47:321–334. 73

[Strunk et al., 2009] Strunk, A., Braun, I., Reichert, S., and Schill, A. (2009). Supporting Rebinding in BPEL. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 864–871, Washington, DC, USA. IEEE Computer Society. 38

[Taylor et al., 2007] Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2007). *Software Architecture: Foundations, Theory and Practice*. Addison-Wesley. 24

[Team, 2000] Team, I. W. S. A. (2000). Web Services architecture overview. IBM developerWorks http://www.ibm.com/developerworks/webservices/library/w-ovr/. 16, 17

[Tigli et al., 2009] Tigli, J.-Y., Lavirotte, S., Rey, G., Hourdin, V., Cheung-Foo-Wo, D., Callegari, E., and Riveill, M. (2009). WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services. *Annals of Telecommunications*, 64:197–214. 50, 51

[von Ammon et al., 2009] von Ammon, R., Ertlmaier, T., Etzion, O., Kofman, A., and Paulus, T. (2009). Integrating complex events for collaborating and dynamically changing business processes. In *Proceedings of the 2009 international conference on Service-oriented computing*, ICSOC/ServiceWave'09, pages 370–384. 33, 53

[Wang and Qian, 2005] Wang, A. J. A. and Qian, K. (2005). *Component-Oriented Programming*. Wiley-Interscience. 24

[WfMC, 1999] WfMC (1999). *Workflow Management Coalition Terminology & Glossary (Document No. WFMC-TC-1011)*. Workflow Management Coalition Specification. 18, 19

[Xiao et al., 2011] Xiao, Z., Cao, D., You, C., and Mei, H. (2011). Towards a Constraint-Based Framework for Dynamic Business Process Adaptation. In *Proceedings of the 2011 IEEE International Conference on Services Computing*, SCC '11, pages 685–692. 46, 49

[Zang et al., 2008] Zang, C., Fan, Y., and Liu, R. (2008). Architecture, implementation and application of complex event processing in enterprise information systems based on RFID. *Information Systems Frontiers*, 10(5):543–553. 2, 140

[Zaplata et al., 2009] Zaplata, S., Kottke, K., Meiners, M., and Lamersdorf, W. (2009). Towards runtime migration of ws-bpel processes. In *Proceedings of the 2009 international conference on Service-oriented computing*, ICSOC/ServiceWave'09, pages 477–487, Berlin, Heidelberg. Springer-Verlag. 22

[Zhai et al., 2008] Zhai, Y., Su, H., and Zhan, S. (2008). A Reflective Framework to Improve the Adaptability of BPEL-based Web Service Composition. In *Services Computing, 2008. SCC '08. IEEE International Conference on*, volume 1, pages 343 –350. 45